

DOCUMENT RESUME

ED 156 157

IB 006 139

AUTHOR Westrom, M. L.
TITLE National Author Language, NATAL-74. Author Guide.
INSTITUTION National Research Council of Canada, Ottawa
(Ontario).
PUB DATE Aug 74
CONTRACT 029-56
NOTE 207p.; Manual was prepared by IEM Canada Ltd.
EDRS PRICE MF-\$0.83 HC-\$11.37 Plus Postage.
DESCRIPTORS Computer Assisted Instruction; *Computers; *Guides;
Manuals; *Performance Specifications; *Programming
Languages; Syntax
IDENTIFIERS Computer Aided Learning

ABSTRACT

The computer language described and illustrated in this manual enables and facilitates the specifications of present and foreseeable required computer operations directly related to the use of computers in instruction. Major sections describe and discuss common features and conventions used in the language, and the three types of language constructs permitted: service functions, units, and procedures. The communication procedures among these three constructs are discussed and examples, where applicable, are presented at the end of each section. (Author/CMV)

* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *

ED156157



National Research
Council Canada

Conseil national
de recherches Canada

ASSOCIATE COMMITTEE ON INSTRUCTIONAL TECHNOLOGY
COMITÉ ASSOCIÉ DE TECHNOLOGIE PÉDAGOGIQUE

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRO-
DUCED EXACTLY AS RECEIVED FROM
THE PERSON OR ORGANIZATION ORIGIN-
ATING IT. POINTS OF VIEW OR OPINIONS
STATED DO NOT NECESSARILY REPRESENT
OFFICIAL NATIONAL INSTITUTE OF
EDUCATION POSITION OR POLICY.

NATIONAL AUTHOR LANGUAGE
NATAL-74

AUTHOR GUIDE

"PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY

Jean-Yves Fortin

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC) AND
USERS OF THE ERIC SYSTEM."

OTTAWA
AUGUST 1974

IR006139

AUTHOR GUIDE

NATIONAL AUTHOR LANGUAGE

NATAL-74

This manual was prepared under
contract No. 029-56 for the
National Research Council of Canada by
IBM Canada Limited

M. L. Westrom
Principal Investigator

August 1974

FOREWORD

This document was prepared under contract by IBM Canada Limited for the National Research Council's Associate Committee on Instructional Technology. It specifies a language to meet the requirements of A Functional Specification for a Programming Language for Computer-Aided Learning Applications previously published by the Committee. The work was guided by a Subcommittee on CAL Programming Languages whose membership is made up of people from across Canada who are actively engaged in a variety of Computer-Aided Learning applications.

The specification at this stage must be considered preliminary and subject to change. Work is currently underway on the first implementation of a language following this specification. As a result of this implementation and subsequent trial applications of the language, revisions can be expected to be made to the specification as it is defined in this manual.

The language specification has been made available for publication at this time in order to solicit critical comment from potential users. Comments and suggestions would be particularly welcome at an early date so that they may be considered in producing the revised specification. Such comments and suggestions should be forwarded to:

The Secretary
Associate Committee on Instructional Technology
National Research Council of Canada
Montreal Road
Ottawa, Canada K1A 0R8

Associate Committee on Instructional Technology

Subcommittee on CAL Programming Languages

J. W. Braham	National Research Council	(Chairman)
G. M. Boyd	Sir George Williams University	
M. A. Brebner	University of Calgary	
B. A. Colpitts	University of Western Ontario	
S. Hunka	University of Alberta	
F. Labrousse	Ministère de l'Education du Québec	
S. K. Lower	Simon Fraser University	
W. P. Olivier	Ontario Institute for Studies in Education	

AUTHOR'S GUIDE

TABLE OF CONTENTS

1.	INTRODUCTION.....	1
1.1	PURPOSE AND SCOPE OF THIS MANUAL.....	1
1.2	PROTOTYPES.....	2
1.3	DEFINITIONS.....	3
1.4	STRUCTURAL ELEMENTS.....	7
1.5	USING THIS MANUAL.....	11
2.	LANGUAGE FORM.....	12
2.1	CHARACTER SET.....	13
2.1.1	Basic Character Set.....	13
2.1.2	Extended Character Set.....	14
2.1.3	Character Classification.....	15
2.2	VARIABLES AND LITERALS.....	17
2.2.1	Switch Variables and Literals.....	17
2.2.2	Numeric Variables and Literals.....	18
2.2.3	String Variables and Literals.....	19
2.2.4	Label Variables and Literals.....	20
2.2.5	Course Variables and Literals.....	20
2.2.6	Vector Variables and Literals.....	20
2.2.7	System Variables.....	22
2.2.7.1	Switch System Variables.....	22
2.2.7.2	Numeric System Variables.....	23
2.2.7.3	String System Variables.....	25
2.3	STATEMENT CONSTRUCTION.....	27
2.3.1	Statements and Comments.....	27
2.3.2	Labels.....	28
2.3.2.1	Internal Labels.....	28
2.3.2.2	External Labels.....	30
2.3.3	Expressions.....	32
2.3.3.1	Switch Expressions.....	32
2.3.3.2	Numeric Expressions.....	34
2.3.3.3	String Expressions.....	34
2.3.3.4	Vector Expressions.....	37
2.3.3.5	Expression Evaluation.....	38
2.3.3.6	Examples of Expressions.....	40
2.3.4	Conversion by Assignment.....	41
2.3.5	Argument Lists.....	45
3.	SERVICE FUNCTIONS.....	48
3.1	FUNCTION STATEMENTS.....	49
3.1.1	The Assignment Statement.....	49
3.1.2	The DO Statement.....	50
3.1.3	The Conditional Statement.....	54
3.1.4	The Null Statement.....	55
3.1.5	The GOTO Statement.....	55
3.1.6	The RETURN Statement.....	55
3.1.7	The END Statement.....	56
3.2	THE GENERAL FUNCTION.....	57
3.2.1	Purpose of the General Function.....	57
3.2.2	Invoking a General Function.....	57
3.2.3	Constructing a General Function.....	58

3.3	THE EDIT FUNCTION.....	60
3.3.1	Purpose of the Edit Function.....	60
3.3.2	Invoking an Edit Function.....	60
3.3.3	Constructing an Edit Function.....	60
3.4	THE GRAPHIC FUNCTION.....	62
3.4.1	Purpose of the Graphic Function.....	62
3.4.2	Invoking a Graphic Function.....	62
3.4.3	Constructing a Graphic Function.....	63
3.4.3.1	The MAP Statement.....	64
3.4.3.2	The PLOT Statement.....	66
3.4.3.3	System-supplied Graphic Functions.....	66
3.5	THE FONT BLOCK.....	71
3.5.1	Purpose of the Font Block.....	71
3.5.2	Invoking a Font Block.....	71
3.5.3	Constructing a Font Block.....	72
3.5.3.1	The CHARACTER Statement.....	73
3.6	SYSTEM FUNCTION AND PSEUDO VARIABLES.....	76
3.6.1	Numeric System Functions.....	76
3.6.2	Numeric Pseudo Variables.....	81
3.6.3	String System Functions.....	81
3.6.4	Pseudo Functions.....	82
3.6.5	String Pseudo Variables.....	85
4.	THE INSTRUCTIONAL UNIT.....	86
4.1	INSTRUCTIONAL UNIT STATEMENTS.....	87
4.1.1	The Unit Header Statement.....	87
4.1.2	The END Statement.....	88
4.1.3	The DISPLAY Statement.....	88
4.1.4	The RESPONSE Statement.....	89
4.1.5	The EDIT Statement.....	92
4.1.6	The Categorization Statement.....	94
4.1.6.1	System-supplied Comparison Functions.....	95
4.1.6.2	Comparison Target Assembly.....	100
4.1.7	The Reprise Statement.....	101
4.2	UNIT CONSTRUCTION.....	103
4.2.1	Order of Execution.....	103
4.2.2	Unit System Variables.....	106
4.3	THE DISPLAY SUB-LANGUAGE.....	107
4.3.1	The Control Command Elements.....	107
4.3.2	Control Command Construction.....	118
4.3.3	Text Conventions.....	119
5.	THE PROCEDURE.....	124
5.1	PROCEDURE STATEMENTS.....	125
5.1.1	The SIGNAL Statement.....	125
5.1.2	The DISABLE Statement.....	126
5.1.3	The ENABLE Statement.....	126
5.1.4	The CHECKPOINT Statement.....	127
5.1.5	The CALL Statement.....	127
5.1.6	The XFER Statement.....	128
5.1.7	The REPEAT Statement.....	128
5.1.8	The STOP Statement.....	129
5.1.9	The EXIT Statement.....	130
5.1.10	The PROCTOR Statement.....	130
5.1.11	The RECORD Statement.....	131

5.1.12	The SET Statement.....	131
5.2	FILE ACCESS.....	134
5.2.1	The Direct-Access Records.....	134
5.2.2	The READ Statement.....	135
5.2.3	The WRITE Statement.....	136
5.3	COURSE RESOURCE CONTROL.....	138
5.3.1	The HOLD Statement.....	140
5.3.2	The RELEASE Statement.....	141
5.4	CONDITION BLOCKS.....	142
5.4.1	The ON CONDITION Statement.....	142
5.4.2	Conditions.....	143
5.4.3	Extended RETURN Statement.....	147
5.4.4	Condition Block Construction and Execution.....	149
5.5	THE GENERAL PROCEDURE.....	153
5.5.1	Purpose of the General Procedure.....	153
5.5.2	Invoking a General Procedure.....	153
5.5.3	Constructing a General Procedure.....	154
5.6	THE ENTRY PROCEDURE.....	157
5.6.1	Purpose of the Entry Procedure.....	157
5.6.2	Invoking an Entry Procedure.....	157
5.6.3	Constructing an Entry Procedure.....	158
5.7	SYSTEM PROCEDURES.....	160
5.7.1	Dynamic Invocation Procedure (LINK).....	160
5.7.2	Student Calculation Mode (CALC).....	160
5.7.3	Compiler & Interpreter Invocation.....	163
5.8	RESTART CONVENTIONS.....	166
5.9	SYNONYM FACILITY.....	167
5.9.1	The SYN Statement.....	167
5.9.2	SYN Examples.....	168

APPENDIX A. LANGUAGE KEYWORDS AND SYMBOLS

Keywords.....	170
Symbols.....	173
Symbols and Keywords by Statement.....	174

APPENDIX B. SYSTEM VARIABLES AND FUNCTIONS

System Switch Variables.....	176
System Numeric Variables.....	176
System String Variables.....	176
System General Functions.....	177
System Edit Functions.....	177
System Graphic Functions.....	178
System Special Functions.....	178
System Procedures.....	178

APPENDIX C. CLASSES OF EXECUTION ERRORS.....179

APPENDIX D. IMPLEMENTATION LEVELS.....181

STATEMENT SUMMARY TABLE.....183

AUTHOR'S GUIDE

INTRODUCTION

1.1 PURPOSE AND SCOPE OF THIS MANUAL

The purpose of this manual is to describe and illustrate the use of a computer language. The syntax and semantics of statements and the conventions for assembly of these statements into coherent specifications of computer operations is presented.

Numerous languages for the specification of computer operations are in existence, each designed for a specific or general purpose. The Language presented in this manual enables and facilitates the specification of computer actions directly related to the use of computers for instruction. The Language is directed solely to this application. Within this application, it is general in scope and attempts to enable and facilitate the specifications of all foreseeable required computer operations as detailed in the National Research Council of Canada Associate Committee on Instructional Technology document "A Functional Specification for a Programming Language for Computer Aided Learning Applications", Ottawa, 1972.

1.2 PROTOTYPES

Throughout this manual, syntactic arrangements are described with reference to a prototype or prototype arrangement. Prototypes may appear on a line by themselves or enclosed in quotation marks. They are to be interpreted according to the following conventions:

1. All alphabetics appearing in upper case represent themselves. These must be entered exactly as they appear.
2. All alphabetics appearing in lower case represent place-holders; these must be replaced by the user with appropriate character strings as defined in the discussion of the construction.
3. All symbols except brackets, and ellipses (three consecutive period characters) represent themselves and these must be entered exactly as they appear.
4. All elements appearing within brackets are optional; that is, they may be included or omitted in the construction.
5. Whenever ellipses appear in a prototype, they are preceded by a component in brackets. This indicates that the element in brackets may be repeated as many times as desired.

1.3 DEFINITIONS

Various terms used throughout this manual require definition because they have, for purposes of this manual, a more restricted or a different meaning from normal usage. Terms requiring extensive explanation are discussed, as appropriate, elsewhere in this manual. The remainder are collected here.

Argument: A placeholder, usually a variable name or expression for a parameter that is to be made available to a routine on invocation.

Author: The person or persons who write computer operation descriptions in the Language.

Component: An assembly of characters; a part of a statement.

Control: Used in the phrases "sequence of control", "execution control", "transfer of control"; Control refers to the determination of which statement will be executed next.

E-notation: One form of representation of a literal numeric. In this form, the numeric appears as "mantissa E exponent". The value is "mantissa" times ten to the power "exponent".

Font: A set of graphics, each of which designates the symbol used to represent a character.

Graphic: A line drawing or a picture composed of discreet points.

Installation: The physical location and/or the group of people in charge of installing and maintaining the language at that location.

Invoke: Commencement of execution of a new routine.

Implementation: The processor or processors and/or the group of people who devise the processors which translate and/or execute the Language.

Keyword: 1. That portion of a statement which identifies it and distinguishes it from other statements. 2. A word designated in the language to be indicative of a specific meaning or action.

Keywords always appear as upper case alphabetic strings in prototypes.

Literal: A component of a statement which at once defines and represents itself. See Variables and Literals, Section 2.2.

Operator: One of the arithmetic operators add, subtract, multiply, divide, or exponentiate or one of the logical operators less than, greater than, equal to, less than or equal to, greater than or equal to, or not equal to. Also the symbols that represent operators, which are (respectively): +, -, *, /, **, <, >, =, <=, >=, and -=.

Parameter: A value passed through argument lists.

Pending: A state of a Procedure, Unit, or Condition Block in which the routine has been invoked, but during execution invoked a lower order routine. The state of waiting for lower order processing to be completed and control returned is called the pending state. A routine in this state is said to be pending. When a routine completes its processing, it becomes inactive and is no longer pending or active.

Routine: 1. Any one of the Entry Procedure, Procedure, Unit, General Function, Edit Function, Graphic Function, Font Block or Condition Block.
2. One or more of these as established by context.

Statement: A sequence of characters having a specific and unambiguous meaning; an element of the Language.

String: Usually a character string; a sequence of characters arranged one after the other.

Student: The person or persons who interact with the computer while the operations as specified by the language are taking place.

System: That part of the implementation which supervises the execution of language specifications.

Vector: Switch or numeric values or place-holders which are considered as belonging to an ordered set.

Word: Special use as established by context:

1. A string of characters containing no space characters; delimited on either side by space characters.
2. A string of digits which may contain one decimal or "E" character delimited on either side by non- numerics or spaces.
3. A string of alphabetic characters delimited on either side by non-alphabetic characters.
4. A string of non-alphabetic characters delimited on either side by alphabetic characters.

In all cases, the end of the string is considered as a word delimiter.

1.4 STRUCTURAL ELEMENTS

The Language is used to prepare computer instructions for the presentation of instructional materials. An assembly of such instructions is called a Course. A Course is composed of a number of routines, and each routine is composed of statements. Some routines may contain statements arranged into DO-Blocks or Condition Blocks. The distinct types of routines that may be constructed are: General Functions, Edit Functions, Graphic Functions, Font Blocks, Units, Procedures, and Entry Procedures. Each of these types of routines are described in detail in following sections.

In defining the instructional specifications, routines may identify and use variables, and may access special "system variables". These variables are of two types, one of which is called simply "variables", the other is called "course variables". The course variables are used in common by all students executing the course so that the author must consider student interaction when accessing them. Normal variables, however, are associated with each particular student executing the routine. The instructional specification is defined as if only one student were executing the routine because each student has a unique copy of each normal variable.

More generally, instructional specifications are composed of only two entities: the Unit and the Procedure. All other routines simply provide means of standardizing and attaining ready access to operations which support and enhance the execution of Units and Procedures.

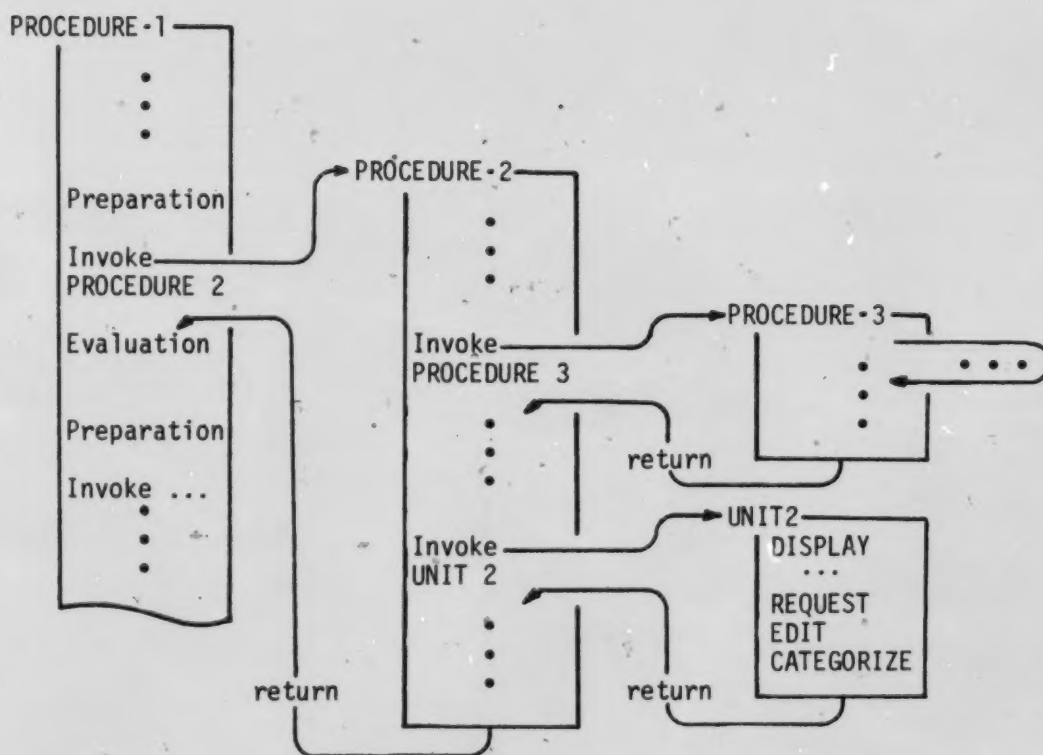
The Instructional Unit provides the immediate interface between the student and the computer. The scope of each Unit is to accomplish a single complete instructional transaction. This typically consists of providing information, asking a question, and then accepting and analyzing the subsequent response.

The Procedure provides means whereby the information required by a Unit can be assembled, the information generated by a student response can be evaluated, and the sequence of instruction can be controlled. Usually, the presentation of a concept requires several instructional transactions. Units are constructed for each allowable transaction and then the task of controlling the presentation of the concept embodied in a single Procedure. Superior Procedures may then be constructed to control the selection and sequence of presentation of these Concept Procedures; and higher-order Procedures constructed to control these.

Each instructional transaction or Unit may be considered as the data to be processed by a controlling Procedure. While the Units are constructed according to a particular strategy, the controlling Procedure exhibits the teaching strategy in its selection and ordering of the Units it invokes. Minor alterations in the controlling Procedure may be employed to examine the ordering and selection strategies or logic of the teaching of a concept. Similarly, the logic of higher-order Procedures (controlling the ordering and selection of concept Procedures) may be examined by experimentation involving only minor modifications.

Now, rather than considering these Units and Procedures as being "locked-in" to a single Course, there is no reason to prevent a number of Courses from accessing them. If a collection of Units and Procedures have been assembled in a pedagogically sound manner, they can be expected to "teach" a concept. An author preparing another Course may simply invoke the highest-order Procedure controlling that assembly and then have some confidence that when control is returned to his Course, his students will "know" that concept.

The above scenario dictates that all routines must exhibit some degree of modularity. That is, each Procedure must be relatively independent of the Procedures that may invoke it. This modularity principle has led to a number of restrictions upon what may and may not be done in the various routines. These are discussed in the following sections.



INVOKING PROCEDURES AND UNITS

1.5 USING THIS MANUAL

This manual is composed of four main sections (excluding the Introduction). Section Two (Language Form) describes features and conventions used in the language which are common throughout. The contents of Section Two are fundamental to the descriptions in the remainder of the manual. The remaining major sections discuss the three types of constructs permitted: Service Functions, Units and Procedures. These sections are, as far as possible, complete in themselves. Where they are not complete, reference is made to the relevant parts of this manual. The communication procedures among these three constructs are discussed in all three sections. Examples, where applicable, are collected at the end of each sub-section.

Routines in the Language are specified by statements. Statements consist of characters which, for some statements, may be grouped into statement components such as labels, keywords, and expressions. These statements control the order in which other statements are executed (sequence control), the allocation and value of variables, and various Input/Output operations.

This section identifies and defines procedures and conventions concerning components of the Language fundamental to the construction of statements. The character set used for construction of statements is specified; variables, literals, labels, expressions and argument lists as components of statements are discussed.

2.1 CHARACTER SET

2.1.1 Basic Character Set

Routines may be specified through use of the following Basic Character

Set: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V,
W, X, Y, Z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and

<u>Character</u>	<u>Name of Character</u>
	space
=	equals
+	plus
-	minus or hyphen
*	asterisk
/	slash
(	left parenthesis
)	right parenthesis
,	comma
.	period or decimal point
:	colon
;	semi-colon
&	ampersand
'	apostrophe
#	number
<	less-than
>	greater-than
_	underscore

2.1.2 Extended Character Set

The Extended Character Set consists of the Basic Character Set plus the characters: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ã, ä, ç, é, ê, ë, ì, ï, ð, ñ, and

<u>Character</u>	<u>Name of Character</u>
!	exclamation
"	quote
\$	dollar
%	per cent
?	question mark

The character sets may be further extended without limit by conventions established both within and without the Language. The Extended Character Set complies with CSA Standard Z243.4 7-Bit Coded Character Sets for Information Processing Interchange.

2.1.3 Character Classification

DIGITS The characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 are classified as digits.

UPPER CASE LETTERS The characters A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z are classified as upper case letters.

LOWER CASE LETTERS The characters a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, ã, ä, ç, é, è, ê, ì, ð, ò, and ù are classified as lower case letters.

ALPHABETICS The union of the upper case letters and the lower case letters are classified as alphabets.

SPACE CHARACTER The space character is used as a delimiter in some statement constructions. In general, wherever one space is permitted or required, one or more spaces may appear. Spaces may be used freely to improve the appearance and readability of the routine.

SYMBOL All characters in the Extended Character Set which are not Alphabets or Digits are called symbols. The following two-character symbols are also used in the language:

`/* */ <- ** <= >= -= << >>`

RESERVED SYMBOL The following characters are reserved symbols: equals, plus, minus, asterisk, slash, left parenthesis, right parenthesis, comma, period, colon, semi-colon, ampersand, apostrophe, number, less than, greater than, and the two-character symbols.

2.2 VARIABLES AND LITERALS

Logical states, numerical values, character strings, and routine locations may be manipulated within routines through the use of variables and literals. Variables are names of storage locations; the information placed in the location associated with a particular variable name may change, but the variable name determines the location and the size of the storage area. Variable names are constructed of alphabetic and digit characters and the underscore symbol; an alphabetic character must be the first character of the name. In the case of label variables and course variables, the variable name must be prefixed with an ampersand or number symbol respectively. Every variable has an associated value type which is either "switch", "numeric", "string", or "label".

Variables of the label type have distinctive names (beginning with an ampersand character). The remaining types are determined dynamically by the context of their use. Literals are not associated with named storage areas, and their values may not be changed. The sequence of characters denoting the literal identifies both the type and the value of the literal.

2.2.1 Switch Variables and Literals

A switch variable is a variable which may assume only two states. One of these states may be named as OFF or FALSE; the other as ON or TRUE. In some cases, the numeric literal "0" may be used to indicate FALSE and the numeric literal "1" used to indicate TRUE. Switch literals TRUE and FALSE are provided for use where numeric literals are not appropriate.

2.2.2 Numeric Variables and Literals

A numeric variable is a variable which may assume an integer or fractional (real) value: positive, negative, or zero. Numeric literals are expressed as a string of digits which may contain a decimal point (period), may be prefixed with a sign (plus or minus), and may be suffixed with "En" (where "n" is a signed or unsigned one-digit or two-digit integer). If a decimal point is not present, the numeric is also called an integer. If "En" is not present, the string of digits represents its normal real value in base ten. If no sign is present, the value is assumed to be positive. If "En" is present, the literal is said to be expressed in E-notation. The value of this string is the normal base ten value of the digit string multiplied by ten to the exponent "n".

For example:

6.3E2 indicates the base 10 value 630

935.63E-2 indicates the base 10 value 9.3563

1E99 indicates the base 10 value 1×10^{99}

1E-99 indicates the base 10 value $1/10^{99}$ or 10^{-99}

2.2.3 String Variables and Literals

A string variable is a variable which may assume the value of a variable-length string of characters. A string having zero length is called the Null string. When the string variable name appears alone, the entire string is referenced. Parts of the string may be referenced by the appending of "(expression)" to the name. For example, NAME(5) refers to the fifth character of the string variable NAME. The structure "(expression)" is called a subscript. The subscript must determine a single numeric integer value which must not be less than or equal to zero, nor greater than the length of the string.

String literals are expressed as the apostrophe symbol followed by a character string and suffixed with the apostrophe symbol. The value of the string literal is the character string without the beginning and ending apostrophes. To represent the apostrophe characters in a character string, two consecutive apostrophes are required. Two consecutive apostrophes are always interpreted as a single apostrophe unless the entire string literal consists of only two apostrophes; in this case, the zero length Null string is indicated. The space character is a valid and significant character in a string literal. Subscripts are not permitted on string literals.

2.2.4 Label Variables and Literals

A label variable is a variable which assumes the value of a location within the routine in which it is used. Label variable names must be prefixed with the ampersand (&) character. If a statement is a labelled statement, then the label portion of that statement is a literal label. Its value is the location of that statement. Labels are discussed in more detail in Section 2.3.2.

2.2.5 Course Variables and Literals

A course variable is a switch, numeric, or string variable which maintains its identity for all students who reference it. Course variables may be of the switch, numeric, or string type. They are distinguished from normal variables in that their names must be prefixed by the number symbol (#). Use of course variables is discussed in Section 5.3.

2.2.6 Vector Variables and Literals

Vectors are one dimensional arrays of either switch elements or numeric elements. They are similar to strings in that each consists of a series of elements which can be referenced as a whole, or individually. A vector of zero length is called a Null vector.

Vectors are named in the same way as switch variables or numeric variables. When the vector name appears alone, the whole vector is indicated but parts of the vector may be referenced through subscripts. For example, $N(5)$ refers to the fifth element of the numeric vector N . The subscripts must be single value integer numeric expressions with value greater than zero and not greater than the length of the vector.

The values of elements of a vector are altered by the same means as any switch or numeric variable. The length of a vector may be changed only when the name appears unsubscripted: on the left side of an assignment statement, in the read-list of a READ statement, or as a variable name in a CALL statement to a Procedure.

Literal vectors are constructed as a list of literal switch or numeric values separated by commas and enclosed in parentheses. In argument lists, where a literal vector may appear with other arguments (separated by commas) the literal vector parentheses are not required if the literal vector is the only parameter of the argument list.

Groups of values, designated by literals or variable names, may be designated as a vector in expressions by enclosing the list of value in parentheses. For example $(3, 2*2, N)$ forms a pseudo vector which exists only for the duration of the statement in which it appears. Pseudo vectors may not contain elements which are strings; all elements must have either switch or numeric values. The

pseudo vector is considered to be of the switch type only if all of its values are either zero or one. Otherwise, it is of numeric type. If N were a vector of length 3 containing (5, 8, 10) then (1, 3, N) would be a pseudo vector of length 5 containing (1, 3, 5, 8, 10). Pseudo vectors must not contain imbedded pseudo vectors.

2.2.7 System Variables

Switch, Numeric, and String system variables are defined. The values any variable may take are those defined for that value type. The names of the system variables are reserved, and may not be used for variable names by the author. Note that further system variables are defined under Pseudo Variables in Section 3.6.

2.2.7.1 Switch System Variables

FRIGHT, FWRONG, FO, F1, ..., F9: At the time of entry or re-entry to any Procedure or Unit, these system switches are all set to FALSE. If, during execution of a Unit, the RIGHT Condition is raised, the FRIGHT system switch is set to TRUE; if the Condition WRONG is raised, FWRONG is set to TRUE; and if any of the FO, F1, ..., F9 Conditions are raised, the system switch of the same name is set to TRUE.

FOTIME: At the time of entry or re-entry to any Procedure or Unit, FOTIME is set to FALSE. If a time limit is specified on a RESPONSE statement and this time limit is exceeded during execution, then the OTIME Condition is raised and the FOTIME system switch is set to TRUE.

FUNREC: At the time of entry or re-entry to any Procedure or Unit, FUNREC is set to TRUE. If any of the Conditions RIGHT, WRONG, FO, ... F9 are raised, FUNREC is set to FALSE.

2.2.7.2 Numeric System Variables

IERR: IERR is normally zero, but if an ERROR Condition is raised, the nature of the error is indicated by an integer value contained in IERR.

IDevice: An index number indicating the device number (serial number) of the default device controlled by the student currently executing the routine.

ITYPE: (mnemonic for Input TYPE): The class of input received from the student: 0=timed out, 1=keyboard response, 2=stylus response; further IYPEs may be established by the installation. ITYPE is set immediately after the execution of every RESPONSE statement.

ICRSE: An index number indicating the order in which students began the course in which the variable is referenced. For the first student, the value of the variable is 1; for the second, 2; and so on.

ISESN: An index number indicating the order in which students began the current session of the particular course in which the variable is referenced. For the first student, the value of the variable is 1; for the second, 2; and so on.

LATENCY: The number of tenth seconds elapsed between the last request for a student response and receipt of that response. LATENCY is set every time a Unit RESPONSE statement is executed.

NRESP: The number of responses entered by the student during execution of the previous unit.

NRIGHT, NWRONG, NFO, NFI, ... NF9: At the time of invocation of any Procedure, a set of these variables - local to the Procedure - are established and initialized to the value zero. If the name of one or more of these variables appears in the argument list of the Header statement of the Procedure, then these are initialized to the value provided by the parameter. These counters are implicitly incremented by one each time one of the corresponding Conditions (RIGHT, WRONG, FO, FI, ..., F9) is raised in any Unit called by the Procedure. Note that because of the Unit RETRY statements, a Condition may be raised more than once during execution of a single Unit. These counters may be modified explicitly by statements in the Procedure.

NOTIME: At the time of entry to a Procedure, NOTIME is set to zero unless otherwise initialized by a parameter passed via the argument list. Each time the OTIME (Overtime) Condition is raised, the counter is incremented by one. Note that the OTIME Condition may be raised more than once during execution of a single Unit. NOTIME may be modified explicitly.

NUNREC: At the time of entry to a Procedure, NUNREC is set to zero unless otherwise initialized by a parameter passed via the argument list. Each time the UNREC Condition is raised, the counter is incremented by one. Note that the UNREC Condition may be raised more than once during execution of a single Unit. NUNREC may be modified explicitly.

PARS: PARS is normally a Null vector. At student registration time however, the registrar may specify a series of numeric values which become the contents of PARS for that particular student. PARS may be altered by assignment statements.

2.2.7.3 String System Variables

RESPONSE: When a Unit RESPONSE statement is executed, the incoming student response is placed in (or appended to) the system variable RESPONSE. For keyboard responses, the received characters are placed in the variable; for graphic response, the index of the row and of the column of the point indicated are placed in the variable in character form separated by a comma character. RESPONSE may not be altered except implicitly by the RESPONSE statement.

REDITED: The Unit EDIT statement causes the student RESPONSE to be edited and the result placed in the system variable REDITED. If no EDIT statement is present in the Unit, the contents of REDITED are the same as those of RESPONSE. REDITED may be altered explicitly.

SID: The student's identification code is contained in the system variable SID. SID may not be altered.

SNAME: The student's usual or "nick" name. The familiar name by which the program would address the student. This is provided in the student's registration. The variable may not be altered.

STUDENT: The student's full name in the form: surname, space, first name, space, initial (if any), as found in the student's registration. This variable may not be altered.

QUEUE: Beginning with the label of the Entry Procedure (the Course name), QUEUE contains the labels of all pending Procedures, pending Condition Blocks, and pending Units. The labels are arranged in the order that the routines were invoked and are separated by a single space.

#QID: #QID is a string Course variable containing the identification codes of all students currently executing the course in which it is referenced. The codes are arranged in the order in which the students began the course and are separated by a single space. The variable may not be altered by the author.

2.3 STATEMENT CONSTRUCTION

2.3.1 Statements and Comments

For specification of routines, the normal line delimiters (such as carriage return) are not usually recognized. (An exception is within the Display statement AS-IS mode.) Rather, the routine definition is accepted as a continuous stream of input consisting of statements and comments. Within this stream, the beginning of a comment is identified by the two-character symbol '/*'; the end of the comment is identified by '*/'. The comment is considered to consist of both the beginning and ending symbol and the character string between them. A comment must not contain an embedded '*/' symbol, but may contain any other characters. Comments are not allowed within the definition of string literals or within the text portion of DISPLAY statements; but otherwise, comments are allowed wherever spaces are allowed.

All parts of routines which are not comments are considered to be statements. Every statement except the Assignment statement is characterized by a keyword. This keyword may be preceded by a label and may be followed by a Statement Body. Every statement has a semi-colon as the last character of the statement or the statement block.

All statements must be coded with the Basic Character Set. If an extended character set is used, the extra symbols appearing in the text portion of Display statements, in string literals, or in comments are accepted. In all other places, lower case alphabets are converted to their corresponding upper case equivalents; all other symbols are deleted from the input stream.

2.3.2 Labels

All labels (except Unit labels) consist of alphabets, digits and the underscore symbol, and begin with one of the alphabets. Every statement may be labelled, but under certain conditions, the label is ignored. All Header statements must be labelled. If a statement is a labelled statement, then the label portion must be at the beginning of the statement and must be delimited from the remainder of the statement by a colon. The colon is not part of the label. Any number of spaces may appear on either side of the colon. The labels on Header statements are called external labels. All other labels are internal labels.

2.3.2.1 Internal Labels

Internal labels are provided to enable the user to associate a logical name with a location in a routine. He may then cause statement execution to begin at that location from other points in the routine. While the DO and Conditional statements may effect sequence of execution without reference to labels, the only means of explicit transfer of control to internal labelled

statements is via the GOTO statement. Within any routine, every statement except the Header statement may have an internal label.

The "location" assigned by the GOTO statement may be either a literal label, an indexed literal label, or a label variable.

The literal label is constructed according to the rules for constructing labels and is associated with a particular statement by being placed on that statement. The association between a literal label and a location cannot be altered during execution of a routine. The literal label is referenced by the statement "GOTO label;" where "label" is an exact replication of the character string used when the label was defined.

The indexed literal label is constructed of two parts: the first is the same as a literal label, the second has the form "#n" where "n" is a literal integer greater than zero. The indexed literal label is formed by the concatenation of these two parts (for example: LOCN#3). It is associated with a fixed location in the same way as a literal label and this association cannot be changed during execution of the routine. The indexed literal label is referenced by the statement "GOTO label#(expression);" where "label" corresponds exactly to the first part of the indexed literal label and where "expression" after evaluation at execution time, produces an integer numeric result equal to the value of "n" in the label definition. For example, if the variable "N" has the value three, the statement "GOTO LOCN#(N);" transfers control to the statement labelled "LOCN#3". This same statement could also be referenced by "GOTO LOCN#3;" - using the indexed literal label as a literal label.

The label variable is constructed in the same way as any variable; however its name must be prefixed with the ampersand symbol. When created, the label variable is not associated with any location in the routine. A location may be placed in a label variable with an assignment statement where the label variable appears on the left hand side and a literal label, indexed literal label (which may require the evaluation of an expression), Null, or another label variable appears on the right. Operations between labels and/or label variables are not permitted.

2.3.2.2. External Labels

External labels are so named because they are "known" outside of the routine in which they are defined. Each routine has exactly one external label associated with it. This is the label that appears on the Header statement of the routine. Routines are invoked by a reference to their external label.

All types of Functions and the Character Definition Block must be labelled with non-indexed literal labels, and may be referenced only by an exact replication of that label. Units and Procedures, however, may have literal labels or indexed literal labels, and are referenced by the Procedure CALL and XFER statements, and the LINK System Procedure. External labels are composed of the same characters as are internal labels, except that every Unit label must be prefixed with the asterisk character.

Whenever an external label is encountered, the invocation or transfer of control to the routine with that label on its Header statement occurs. General Functions are invoked by the appearance of their label in an expression. Edit Functions are invoked by the appearance of their label in an EDIT statement. Graphic Functions are invoked by the appearance of their label in an "&G" control element on a DISPLAY statement or in a PLOT statement. Character definitions, or Font Blocks are invoked by the appearance of their label in an "&F" control element in a DISPLAY statement. Units and Procedures are invoked by the appearance of their label in a Procedure CALL or XFER statement or indirectly through the system Procedure LINK.

Units and Procedures are the only types of routines which may have indexed labels. External indexed labels are constructed in the same way as internal indexed literal labels. Units and Procedures are invoked by the statement "CALL name(argument-list);". Here, "name" may be either an external label or an external indexed label. Label variables are not used. If a Procedure has been labelled "TEST#2" and if the variable N has the value two in the current Procedure, then the external Procedure may be invoked by either "CALL TEST#2(arg-List);" or "CALL TEST#(N)(arg-list);". If the external label is not indexed, then an exact replication of the label must be used to invoke its associated Procedure.

2.3.3 Expressions

Within the Language, variables and literals are arranged along with arithmetic operators, relational operators, and General Function references to produce expressions. Variables and literals have already been discussed. The arithmetic operators are denoted by the plus, minus, asterisk, slash and double asterisk symbols. The results produced by the presence of these operators depends upon the variables or literals to which they are applied; but in general, they represent the addition, subtraction, multiplication, division and exponentiation operations respectively. The relational operators are < (less than), > (greater than), <= (less than or equal), >= (greater than or equal), = (equal), and != (not equal). These are sometimes called Logical operators because the result of any of them applied to two variables always produces a Logical (TRUE or FALSE) result. Spaces may appear on either side of any operator or operand.

2.3.3.1 Switch Expressions

The arithmetic operators plus and minus may appear as prefix operators on switch variables. The plus sign leaves the value unchanged; minus indicates boolean negation. Note that the representation "-1" is taken as the numeric literal rather than a boolean negation of the switch value. Switch expressions may be composed of relational expressions, switch variables, switch vectors, the numeric literals 0 and 1, and switch General Function references. The action of infix operators between single switch values are as follows:

- switch + switch: Result is an Inclusive OR, ie:

TRUE + TRUE is TRUE	FALSE + TRUE is TRUE
TRUE + FALSE is TRUE	FALSE + FALSE is FALSE
- switch - switch: Result is an Exclusive OR, ie:

TRUE - TRUE is FALSE	FALSE - TRUE is TRUE
TRUE - FALSE is TRUE	FALSE - FALSE is FALSE
- switch * switch: Result is an AND, ie:

TRUE * TRUE is TRUE	FALSE * TRUE is FALSE
TRUE * FALSE is FALSE	FALSE * FALSE is FALSE
- switch / switch: Result is a NOR, ie:

TRUE / TRUE is FALSE	FALSE / TRUE is FALSE
TRUE / FALSE is FALSE	FALSE / FALSE is TRUE
- switch ** switch: Result is an IMPLICATION, ie:

TRUE ** TRUE is TRUE	FALSE ** TRUE is TRUE
TRUE ** FALSE is FALSE	FALSE ** FALSE is TRUE

When relational operators are applied between switches, the switches act as their numeric equivalent values of zero and one. If the relational expression is TRUE, the result is one, otherwise the result is FALSE or zero. For example,

TRUE = TRUE is TRUE	TRUE < FALSE is FALSE
TRUE == TRUE is FALSE	TRUE > FALSE is TRUE

All expressions containing relational operators are relational expressions, have a switch result (TRUE or FALSE), and are also switch expressions.

2.3.3.2 Numeric Expressions

The plus and minus symbols may appear as prefix operators on numerics. Minus reverses the sign of the numeric, plus has no effect. As infix operators, plus, minus, asterisk, slash and double asterisk produce a result established by addition, subtraction, multiplication, division or exponentiation respectively. Roots may be obtained by exponentiation to a fraction. Numeric expressions are composed of switch expressions and numeric literals, variables, vectors, and/or Function references. The result of a numeric expression is always numeric. Whenever a switch is involved in an operation with a numeric, the switch is converted to numeric one or zero before the operation is performed. Relational operators between numerics produce a switch result TRUE (1) if the relationship is true, FALSE (0) otherwise.

2.3.3.3 String Expressions

Plus and minus must not appear as prefix operators on strings. Switch and numeric expressions and string variables or literals may appear in string expressions. Whenever a switch takes part in an operation with a string, the switch is first converted to numeric one or zero and behaves in the same way as a numeric. Double asterisk or the exponentiation operator must not be used with strings.

In most operations between numerics and strings, the numeric is converted to a character string in that operation. The length of the character string produced depends upon the numeric, but in general, it is as short as possible. If the numeric is an integer, then only those characters necessary to represent that integer are produced. If a non-integer value, then up to three places of decimal are produced.

The defined operations between strings and among strings and numerics are presented in the following tables. The CONVERT entry indicates that the numeric is converted to a string just before performing the indicated operation. Subscripts on the table entries refer to following notes:

PLUS	NUMERIC ₂	STRING ₂
NUMERIC ₁ +	NORMAL SUM	CONVERT
STRING ₁ +	CONVERT	CONCATENATE ₁
MINUS	NUMERIC ₂	STRING ₂
NUMERIC ₁ -	NORMAL DIFFERENCE	CONVERT
STRING ₁ -	CONVERT	FIRST OCCURENCE DELETION ₂
ASTERISK	NUMERIC ₂	STRING ₂
NUMERIC ₁ *	NORMAL PRODUCT	REPLICATION ₃
STRING ₁ *	REPLICATION ₃	CONCATENATE WITH INTERVENING SPACE ₁
SLASH	NUMERIC ₂	STRING ₂
NUMERIC ₁ /	NORMAL QUOTIENT	CONVERT
STRING ₁ /	CONVERT	ALL OCCURENCE DELETION ₄

NOTES

1. **CONCATENATE:** The result produced is a string consisting of a concatenation of the two operand strings; string₁ followed by string₂.
2. **FIRST OCCURENCE DELETION:** String₁ is scanned for an occurrence of string₂ within it. If one is not found, the result is string₁. If one is found, the result is string₁ with the first occurrence of string₂ removed.
3. **REPLICATION:** The result is string concatenated to itself numeric times. Numeric is truncated if it is not an integer.
4. **ALL OCCURENCE DELETION:** The result consists of string₁ with all occurrences of string₂ deleted.

The relationship operators applied between strings compare the strings on a character for character basis, from left to right. If a difference is found, the string containing the character higher in the collating sequence at that point is the greater. If the strings are of unequal length and no difference is found within the characters of the shorter string, the longer string is greater. Strings are equal only if they have the same characters and the same length. The collating sequence of characters is determined by the value of the system SET variable SCOL.

2.3.3.4 Vector Expressions

Numeric and switch vectors may participate in expressions. As a prefix operator, a minus sign changes the sign of all numeric values in a vector. The entire vector is indicated when its name appears without a subscript. If two vectors are of the same length, an infix operator applied between them produces a resulting vector of the same length. Each element of the resulting vector is produced by applying the infix operator to the corresponding elements of the operand vectors. Strings may not participate in vector operations, nor may vectors of differing lengths.

If an infix operator appears between a vector and a switch or numeric, the result is a vector of the same length as the operand vector; each element is the resultant of the infix operator applied between the single element and the corresponding operand vector element. For example, "6/(1,2,3)" yields the result "(6,3,2)".

2.3.3.5 Expression Evaluation

An expression identifies a set of variables and literals which are to be combined according to the rules established for the prefix and infix operators. In the evaluation of an expression, the actual calculations must be done serially; each prefix operator being applied to its single variable and each infix operator being applied to its two operands. The arrangement of the variables and the operators affects the order of calculation and must be considered in the construction of an expression. The rules for expression evaluation are:

1. All Function references are replaced by the value returned; the Functions are invoked from left to right within parentheses.
2. All exponentiation (including logical IMPLICATION) operations are performed, and prefix operators applied from right to left within parentheses.
3. All multiplication and division (including AND and NOR) operations are performed, proceeding from left to right within parentheses.
4. All addition and subtraction (including OR) operations are performed, proceeding from left to right within parentheses.
5. All relational operations are performed, proceeding from left to right within parentheses.

In this process, evaluations are always performed within parentheses first so that parentheses may be used to indicate the desired order of calculation. If a set of parentheses encloses a list of switch or numeric values then this is taken as a vector in the remaining calculations.

Expressions are categorized according to the type of result produced after evaluation. Switch expressions produce a switch result, numeric expressions, a numeric result, string expressions a string result, and vector expressions produce a vector result. Since expressions may appear within expressions, and since some operations are not permitted between certain value types, the type of each sub-expression must be considered by the author when constructing the overall expression.

2.3.3.6 Examples of Expressions

Switch Expressions

1. A1 - 1
2. N3 < 12.5
3. A1 + 0 + (NUM=6)
4. BR*FUNC (N,3,'ABC')

Numeric Expressions

1. N**2
2. (-NB+(NB**2-4*NA*NC)**.5)/2*NA
3. 12 + (INC<3) ...see note 1
4. INDEX(RESP,'?')

String Expressions

1. 'This is a literal string'
2. ''
3. STR + 2*N - A3 ...see note 2
4. REP/SUBSTR(REP,1,3)

Vector Expressions

1. (1,0,1,1,1)
2. 3 * (3,4,5)/11
3. (1,4,6,8) **2 - (6,6,6,3)
4. NUMBR(RESPONSE)

Notes

1. If the parentheses were omitted, this would be a switch expression evaluated as (12 + INC) < (3).
2. In the evaluation of this expression, 2*N is converted to a character string and concatenated to STR; then the first character zero or one (depending on the value of A3) is deleted from the string.

2.3.4 Conversion by Assignment

As already discussed, values may be converted from one form to another during the evaluation of expressions. Conversion may also be required in the assignment of values in Assignment statement, in the passing of parameters via argument lists, and in READ operations. When an obtained value of one type is assigned to a location of a different type, the conversion made is called a conversion by assignment. Label variables may not participate in any form of conversion.

If the destination for a value is a single unsubscripted variable name, then this name is associated with the new value regardless of its previous value. For example, suppose we had `NUM <- 5`. Here, NUM is a single-value numeric. If we now had `NUM <- 'five'`, then the name NUM would lose its association with a numeric location containing value 5 and assume an association with a string location of value 'five'. In all cases, where the destination of a value is expressed as a single unsubscripted variable name, the type of the location is converted and the value is not converted.

If the destination location is expressed as a subscripted variable name, then the type of the location cannot be changed and the value must be converted. For example, suppose the switch vector FLAG contained (1, 0, 1, 1). Then the assignment `FLAG(3) <- 6` requires that the numeric value (6) be placed in a switch location FLAG(3). The rules for this type of conversion are:

1. Switch to numeric: The numeric takes the value 1 if the switch value was TRUE, 0 otherwise.
2. Switch to string: The string character position takes the value '1' if the switch value is TRUE, '0' otherwise.
3. Numeric to switch: The switch takes the value 1 if the numeric was odd; 0 if it was even.
4. Numeric to string: NOT PERMITTED.
5. String to switch: The switch is given value 0 if the string is Null; 1 otherwise.
6. String to numeric: The string is scanned for the first occurrence of a valid literal numeric. If one is found, it is converted and the value placed in the numeric. If one is not found, the value 0 is placed in the numeric.
7. Vector to switch: NOT PERMITTED.
8. Vector to numeric: NOT PERMITTED.
9. Vector to string: NOT PERMITTED.

Pseudo vectors may also be the destination for one or several values. In this case, the pseudo vector elements must all be names of single-value numeric locations. They may not be expressions or names of string locations. One or more values may be assigned to the pseudo vector. If one value is assigned, then all elements of the pseudo vector are set to that value. If more than one value is assigned, then these values are assigned to the pseudo vector on an ordered one for one basis until:

1. The list of assigned values is exhausted. In this case, the list of values is repeated as many times as required.
2. The elements of the pseudo vector are all filled. In this case, the remaining assigned values are ignored.

Examples:

(A,B,C) <-0	Numerics A,B, and C are set to zero.
(A,B,C,D,E) <-(1,2)	A and B are set to 1 and 2 (respectively)
	C and D are set to 1 and 2 (respectively)
	E is set to 1
(A,B) <-(1,2,3,4,5)	A is set to 1; B to 2. The values 3,4, and 5 are ignored.
(A(3),B,C) <-(1,2,3)	A(3) is set to 1; B to 2, and C to 3. (A must be a numeric vector)

The ERROR Condition is raised if a pseudo vector is assigned the Null value or any of the elements are subscripted and do not name numeric vectors.

Examples:

a. LHS <- 'the new value is' +5

Here, there are two operands in the source expression, the string 'the new value is', and the numeric 5. According to the rule for addition of a string and a numeric, the string result 'the new value is 5' is produced. The LHS value is set to the string type and this value.

b. `LHS <- 10 * 'No '`

The right hand side expression after evaluation produces the 30 character string "No No No No No No No No No". LHS is set to the type string and this value.

c. `LHS <- 3 + IOTA(5)`

Assuming the function `IOTA(5)` produced the vector (1,2,3,4,5), LHS is set to numeric vector type with value (4,5,6,7,8) (and length 5).

d. `LHS <- (1,2,3,LHS,9,10)`

Assuming LHS had the value assigned in example "c" above, the pseudo vector on the right hand side is of length 10 with value (1,2,3,4,5,6,7,8,9,10). LHS is set to a numeric vector of this value.

e. `LHS <- 'Please ' + ('do'*SW + 'omit'*-SW) + ' the following:'`

Assuming SW were a switch, evaluation of the right hand side results in the string 'Please do the following:' if SW has value TRUE, and 'Please omit the following:' if SW is FALSE. The parentheses are not required, but are used here to improve readability.

f. If N were a vector of length 3 containing the elements 2, 4, 6, then:

`N(4) <- 8;` is invalid,

`N <- 0;` results in N having a single element of value 0,

`N <- (N,8)` results in N having 4 elements: 2,4,6,8.

g. S is a string variable with initial value 'John':

S(3) <- 'a' changes S to 'Joan'

S(1) <- 5 is invalid

S <- 5 changes S to type numeric with value 5.

2.3.5 Argument Lists

Argument lists are the means of communication between a routine which invokes a subordinate routine and the subordinate routine. In the calling routine, the argument list appears enclosed in parentheses immediately following the name of the called routine. In the called routine, the argument list appears enclosed in parentheses as the last item of the Header statement.

In the called routine, the argument list consists of a series of unsubscripted variable names separated by commas. In the calling routine, the argument list identifies the parameters to be passed by a series of expressions separated by commas. In the calling process, the expressions in the argument list of the calling routine are evaluated (if possible). The parameters (values) obtained are then assigned on an ordered one for one basis to the variable names in the called routine argument list.

Vectors may be passed using vector names and/or pseudo vectors.

The arguments in the calling routine argument list may be of three types.

1. Input Arguments. Input arguments are arguments via which parameters are passed to a called routine but through which values are not

returned to the calling routine. Any argument expressed as a literal, a Course variable, an expression involving at least one operator, a Function reference, or a pseudo vector containing an input argument or a vector name must be an Input argument.

2. Output Arguments. Output arguments do not contain a defined value before invocation of the called routine. The called routine defines these arguments before returning. Output arguments must be expressed as unsubscripted variable names. They may not be used in Function calls.
3. Two-way Arguments. Two-way arguments pass a value into a routine. This value may be modified by the called routine and this modification is maintained when the calling routine regains control. Two-way arguments must be unsubscripted variable names. The type and length of two-way arguments may be altered by the called routine. Two-way arguments may not be used in Function calls.

If fewer arguments are supplied by the calling routine than are required by the called routine, the parameters are assigned on an ordered one-for-one basis until the parameters have been exhausted. The remaining arguments are set to the default value of the Null vector. Parameters may be defaulted in the calling routine argument list by omitting an expression for that parameter, but including the comma delimiter. Elements of pseudo vector arguments may not be defaulted in this way. Label arguments may not be passed via argument lists. Attempting to do so raises the ERROR Condition.

Examples

A Procedure has the Header statement:

```
ABC:PROCEDURE (STR,PAR)
```

If invoked by:

```
CALL ABC('John', 5);
```

the Procedure begins with the string STR initialized with the value 'John' and the numeric PAR initialized at 5.

If invoked by:

```
CALL ABC('John', (6, 5.3, 2*2));
```

the Procedure begins with STR initialized as 'John' and PAR would be a vector of length 3 with values 6, 5.3, and 4.

If N is a vector containing the values (8, 10), and the Procedure is invoked by:

```
CALL ABC('John', N)
```

then PAR is initialized as a vector of length 2 containing the values 8 and 10.

One of the structural elements of the Language is the Function. Four distinct types of Functions are definable within the Language. At the time a Function is defined, its type is determined by the Function Header statement. At the time of invocation, the type is determined by the context of the invocation.

A Function is invoked by the appearance of the label of the Function within a statement of another routine. Only General Functions may invoke themselves. The time that the Function is invoked, within the execution of the statement, is determined by the rules for execution of the statement. The result (explicit or implicit) of the Function is made available for further processing as soon as the Function has executed. If an author defines a Function with the same name as a system Function, then the author's Function takes precedence and he may not invoke the system Function of the same name.

The four types of Functions permitted within the language are: the General Function, the Edit Function, the Graphic Function, and the Font Block. These are described, in that order, in following sections. The statements that are permitted in more than one of these Functions are presented first, and are referenced thereafter as necessary. These statements are also permitted within Procedures and are referenced by the discussion of Procedures.

3.1 FUNCTION STATEMENTS

The statements defined in this section may appear in any Function or Procedure. The additional statements that may be used with particular types of Functions and Procedures are presented with the discussion of the routines.

3.1.1 The Assignment Statement

l.h.s. <- r.h.s.;

The Assignment statement is the only statement which does not begin with a keyword. The "l.h.s." (left-hand side) of the Assignment statement must be a variable name, a subscripted variable name, or a pseudo vector containing only subscripted or unsubscripted variable names. The assignment symbol "<-" identifies the assignment statement; it is composed of the two symbols less-than and minus with no intervening space. The assignment symbol may be preceded and/or followed by zero or more spaces. The "r.h.s." (right-hand side) of the statement is an expression (which may be a vector).

If the left hand side of the assignment statement is a single variable and the right side a non-vector expression, the expression is evaluated according to the rules for arithmetic operations and placed in the variable.

Any assignment statement may contain references to General Functions on the "r.h.s." The entire "r.h.s." is always evaluated before the "l.h.s."

is set with the result. For example, the assignment statement "N<-N+1" will result in the value of N being incremented by one.

Examples

1. A <- B;
2. A <- B + C;
3. (A, B) <- C;
4. (A, B) <- (C, D);

Examples illustrating the evaluation of "r.h.s." expressions appear in Section 2.3.3. Examples of Assignment statements requiring conversions appear in Section 2.3.4.

3.1.2 The DO Statement

DO [variable=spec[,spec]...] [WHILE (logical-expression)];

The DO statement is the Header statement for a DO-Block. A DO-Block consists of a DO statement followed by any number of acceptable statements and/or DO-Blocks, followed by an END statement. The END statement that ends a particular DO Block is called the corresponding END statement. If the DO statement is labelled, an END statement referencing that label ENDS any intermediate DO blocks and becomes the corresponding END statement. There are three forms of the DO statement which imply different forms of execution of the DO-Block.

1. DO; In its simplest form, the DO statement consists of only the Keyword. This implies that the remainder of the statements of the DO-Block are to be executed in turn until control is transferred out of the DO-Block or the corresponding END statement is encountered (whichever occurs first).

2. DO WHILE (logical-expression); In this form, the DO statement causes the logical expression to be evaluated. If the result is TRUE, the statements of the DO-Block are executed consecutively until control is transferred out of the DO-Block or the corresponding END statement is encountered. If the corresponding END statement is encountered, then control is transferred back to the DO-Block Header statement. The logical expression is re-evaluated, and if TRUE, the DO-Block is executed again. This continues until the logical-expression is FALSE at the time it is evaluated or control is transferred out of the DO-Block.
3. DO variable=specification₁ [,specification₂]...[WHILE (logical-expression)]; In this form, "variable" must be a single variable (which may be subscripted) or a pseudo-variable. It may not be a course variable. "Specification" may take the following forms:
 - a. exp₁
 - b. exp₁ TO exp₂
 - c. exp₁ TO exp₂ BY exp₃

In this third form of the DO statement, execution of the DO-Block and the value of "variable" are as follows:

1. If "specification" is a single expression, the DO-Block statements are executed as under the first form (DO;). "Variable" has a value equal to the result of the evaluation of "exp₁" during the execution.
2. If "specification" is in the form "exp₁ TO exp₂", both "exp₁" and "exp₂" are evaluated; "variable" is set to "exp₁", and then compared to the value of "exp₂". If "variable" is greater than "exp₂" then control automatically transfers to the statement following the DO-Block END statement. Otherwise the statements of the DO-Block are executed.

If the corresponding END statement is encountered, control returns to the DO-Block Header; "variable" is incremented by one and again compared to the result of the evaluation of "exp₂". The DO-Block statements are executed repeatedly until either control transfers out of the DO-Block or "variable" is incremented to a value larger than "exp₂".

3. If "specification" is in the form "exp₁ TO exp₂ BY exp₃" then the DO-Block is executed in the same way as for case 2 above except that "variable" is incremented by the value of "exp₃" for each iteration. If the value of "exp₃" is negative, then iteration continues until the value of "variable" is less than "exp₂".
4. If the DO statement includes "WHILE (logical-expression)" then each time control is passed to the DO-Block Header, "logical-expression" is evaluated. The Block is executed only if the result is TRUE. If the result is FALSE, the entire DO-Block is immediately ended.

If more than one specification is present on the DO statement, each specification controls the iteration of the DO-Block and the value of "variable" in turn until the conditions of the specification have been satisfied. Iteration may then continue and the value of "variable" be determined by the specification to the right of the one just completed. If more than one specification is present, they must be separated by commas. The keywords "TO", "BY", and "WHILE" must be preceded by at least one space character.

The "variable" is a normal variable which maintains its value after control has left the DO-Block. When completion of a DO-Block is caused by a FALSE expression in a WHILE clause or from exhaustion of the "specifications" of a case 3 DO-Block, "variable" has a value greater than "exp₂" if "exp₃" was positive or less than "exp₂" if "exp₃" was negative.

The statements in the DO-Block may be labelled statements, and branching may occur within the Block. However, a DO-Block must not be entered at any point other than the DO-Block Header statement.

3.1.3 The Conditional Statement

IF logical-expression THEN group₁ [ELSE group₂]

The Conditional statement designates that "group₁" is to be executed only if "logical-expression" is TRUE, and that "group₂" is to be executed only if "logical-expression" is FALSE. "Group₁" and "group₂" may be the Null statement (a single semi-colon), any permitted single statement (except the DO, END, or a HEADER statement), or a DO-Block. Conditional statements may be nested; that is, either "group₁" or "group₂" may contain a Conditional Statement. Since each "ELSE group₂" clause is always associated with the inner-most unmatched IF in the same Block, an ELSE with a NULL statement may be required to specify the desired sequence of control. Note that the IF statement is not itself terminated by a semi-colon. However, each "group" (if present) must contain at least one statement which is terminated by a semi-colon.

Examples

1. IF A+B THEN CALL *UNIT; (A and B are of type switch)
- 2 IF (N**2=I+J) THEN DO;
 (I,J)<-(I,J)+1;
 GO TO TRY_AGAIN;
 END;
 ELSE DO K=I TO J:
 IND(K)<-NUM(K);
 END;
3. IF N<5 THEN
 IF I-=0 THEN CALL *UNIT;
 ELSE;
 ELSE CALL PROC3(PAR1,PAR2);

3.1.4 The Null Statement

The Null statement consists of one character, the semi-colon. It does not by itself, cause any action but may be required (as described under the Conditional Statement) for specification of sequence of control.

3.1.5 The GOTO Statement

GOTO location;

The GOTO statement causes unconditional transfer of sequence of execution to the position identified by "location". "Location" must identify a location within the routine in which it appears. It may be either a literal label, an indexed literal label, or a label variable.

One or more spaces may appear between the "O" and "T" of the keyword, as: "GO TO location;" A label variable may be assigned the Null value. If such is the case, a "GO TO that-label-variable" statement causes control to continue with the statement following the GOTO statement.

3.1.6 The RETURN Statement

RETURN;

At execution time, there are two means by which the logical end of processing for a particular Function or Procedure may be indicated.

One is the encounter of the END statement corresponding to the Function or Procedure Header statement, the other is the encounter of a RETURN statement.

Functions may be invoked only by the execution of other Procedures or Functions. In a Function, the RETURN statement causes the result of the Function to be made available to the calling routine and termination of further processing within that Function. Encountered in a Procedure, the RETURN statement causes control to revert to the statement in the calling Procedure immediately after the one invoking the current Procedure. In an Entry Procedure, encounter of the RETURN causes the ENDCOURSE condition to be raised. Unless this is trapped by an ON CONDITION in the Entry Procedure, execution of the Course is then terminated.

3.1.7 The END Statement

END [location];

The END statement is used as a statement delimiter to mark the last statement in a DO-Block or routine. If "location" appears on an END statement, it must be either a literal label or a literal indexed label; it terminates the DO-Block or routine headed by that label and all other (if any) open intermediate DO-Blocks. If a label is not present, the END terminates the Block headed by the last preceeding DO or Header statement. If execution control reaches an END statement for a routine, it is treated as a RETURN statement. The END statement that marks the physical end of a Block or routine is called, in relation to the Header, the corresponding END statement.

3.2 THE GENERAL FUNCTION

3.2.1 Purpose of the General Function

The General Function is also called the Arithmetic Function. It allows the user to define a general purpose routine which may access a number of variables to produce a single explicit result.

3.2.2 Invoking a General Function

The General Function is invoked by the appearance of the external label of the Function in a statement of another routine. The Function must have at least one argument in its argument list. Function references are distinguished from subscripted variables by the assumption that variables must be initialized before they are used. Variables are assumed to be initialized if their name appears in the argument list of the Routine Header statement, on the left side of an assignment statement, in a READ statement, or in the argument list of the invocation of a Procedure. An undefined variable may not be referenced by a subscripted name. Users may not define variables of the same name as Functions which they wish to reference in a Routine and must ensure that all strings and vectors are explicitly initialized.

3.2.3 Constructing a General Function

The General Function may employ any of the statements discussed under Section 3.1 (Assignment, DO, Conditional, Null, GOTO, RETURN, and END), the EDIT, the READ and the ON CONDITION statements discussed under Sections 4.1.5, 5.2.2, and 5.3.1 respectively. The General Function must begin with a Header statement and end with a corresponding END statement. The Header statement of the General Function is as follows:

```
label: FUNCTION (argument-list);
```

The General Function Header statement must have a label, which is not an indexed label. It becomes the name by which the Function is referenced externally. A variable of the same name as the label is automatically created. This variable should be set to a value during execution of the Function; this is the value returned to the calling routine. At invocation of the Function, this variable is set to the Null numeric vector, and if unchanged this will be the value returned.

The syntax rules and rationale for construction of the "argument-list" are discussed in Section 2.3.5. Although the values of the variables represented in the argument-list may be altered during execution of the Function, such alterations do not affect the corresponding variables of the calling Routine. Similarly, the results of the EDIT statement are local to the General Function in which it is used and do not affect a change to REDITED once the Function has completed.

General Function Example

```

IOTA: FUNCTION(N);
/*
 * THIS FUNCTION SIMULATES THE APL MONADIC IOTA *
 * OPERATOR. IF N HAS A SINGLE VALUE, A VECTOR IS *
 * RETURNED CONTAINING THE INTEGERS 1 THRU N. IF *
 * N CONTAINS 2 VALUES, THESE ACT AS UPPER AND *
 * LOWER LIMITS ON THE INTEGER VECTOR RESULT. *
 */
IOTA <-NULL;
IF (LEN(N)<=0)THEN RETURN;
IF (LEN(N) =1)THEN
    IF(N<=0) THEN RETURN;
    ELSE
        DO I = 1 TO N;
        IOTA <- (IOTA,I);
        END;
ELSE
    DO I = N(1) TO N(2);
    IOTA <-(IOTA,I);
    END;
END;

```

Use of the IOTA Function

1. INT <- IOTA(100);
setting the vector INT to the integers 1, 2, 3,..., 100.
2. INT <- IOTA(5,10)
setting INT to 5,6,7,8,9,10.

3.3 THE EDIT FUNCTION

3.3.1 Purpose of the Edit Function

The EDIT statement in a Unit permits the user to specify editing to be performed on the REDITED system string variable. Edit Functions do this editing. A number of built-in functions are provided for this task: these are discussed in Section 4.1.5. The author may design his own further editing functions as discussed in this section.

3.3.2 Invoking an Edit Function

Edit Functions are invoked by the appearance of the external label of the Edit Function in the list of references on an EDIT statement. The Function reference may not pass parameters; i.e., no argument list is permitted. Neither does the Edit Function return an explicit value. Rather, the actions of the Function are implicitly supplied within the REDITED string variable. The Edit Functions specified on an EDIT statement are executed in the order (from left to right) in which they appear.

3.3.3 Constructing an Edit Function

The Edit Function may employ any of the statements discussed in Section 3.1 (Assignment DO, Conditional, Null, GOTO, RETURN, and END), and the EDIT statement discussed in Section 4.2.5.

The Edit Function must begin with a Header statement and end with an END statement. The syntax of the Header statement is as follows:

```
label: EDIT FUNCTION;
```

The Edit Function Header statement must have a label which is not an indexed label. It becomes the name by which the Function is referenced. An Edit Function may not invoke itself. The purpose of the Edit Function is to alter the contents of REDITED based on information obtained from system variables (usually RESPONSE and REDITED). The actions of the EDIT statement within an Edit Function are not local to the function.

Edit Function Example

```
NUMSCAN: EDIT FUNCTION;
/* THIS EDIT FUNCTION CONVERTS SPELLINGS OF *
* NUMBERS 0 THRU 9 TO THEIR NUMERIC *
* EQUIVALENTS *
* */
EDIT SHUP,MULSP;
I <-NUMSTR(REDITED,'ONE');
IF (I=0) THEN
    DO J=1 TO I;
    SUBSTR(REDITED, INDEX(REDITED,'ONE',3))<- '1';
    END;
I <- NUMSTR(REDITED,'TWO');
.
.
IF (I=0) THEN
    DO J=1 TO I;
    SUBSTR(REDITED, INDEX(REDITED,'NINE'),4) <- '9 ' ;
    END;
EDIT MULSP;
END;
```

This sample Edit Function is invoked by the appearance of its name in the edit-list of an EDIT statement in a Unit; as

```
EDIT NUMSCAN,NUMBR;
```


3.4 THE GRAPHIC FUNCTION

3.4.1 Purpose of the Graphic Function

For those users who have devices which permit the display of graphics, Graphic Functions are provided to permit the construction and display of drawings.

3.4.2 Invoking a Graphic Function

Graphic Functions may be invoked within other Graphic Functions by the PLOT statement, within Font Blocks with the CHARACTER statement, and within Units via the &G control element of the Display Sub-language. The syntax of the control element is as follows:

`&G(name1(arg-list1) [,name2(arg-list2)])...`

This element causes the named Graphic Functions to be invoked in order of appearance (left to right). Each Function displays immediately the result of its processing (if any). For every Function invocation, the location of the next print position (cursor) at the time of the invocation is taken as the origin (0,0). The Graphic Functions (except CHAR) do not do any automatic display erasure. It is therefore reasonable for the user to erase the area in which the generated graphic will appear and position the cursor at the lower left hand corner of this area before invoking the Graphic Function(s). Point Graphics do not affect cursor position, but vector graphics set the cursor at the character position containing the pen as left on completion of the Vector Graphic.

3.4.3 Constructing a Graphic Function

The Graphic Function may use any of the General Function statements Assignment, DO, Conditional, Null, GOTO, RETURN, and END; as well as the PLOT and MAP statements. The PLOT statement allows a Graphic Function to invoke another Graphic Function, and enables graphing by the drawing of lines. The MAP statement enables graphing by the "placement" of points (of light) on the graphic device.

In the display of characters on a graphic device, coordinates of position on the screen are expressed in character lines (counting from the top of the screen, the first line is number 1) and character positions (counting from the left of the screen, the first position is number 1). However, each character is considered to be composed of an array of "dots" and these dots may be considered as arranged in rows and columns on the graphic device. Within Graphic Functions, coordinates are always expressed in dot rows (counting from the bottom row which is number zero) and dot columns (counting from the left-most column which is number zero). Wherever a specific position on the screen is to be indicated, this is done by naming the x-coordinate and the y-coordinate (in that order) of the particular dot required. The dot at position (0, 0) is called the origin dot or origin.

Two means are provided for producing graphics. One is through the specification of which of the dots are to be ON. This is called a POINT Graphic and is accomplished using the MAP statement. The other is specifying the beginning and end points of lines to be drawn; this is called a VECTOR Graphic and is accomplished by the use of a number of elemental system-supplied Graphic Functions.

The Graphic Function must begin with a Header Statement and end with a corresponding END statement. The Header statement is as follows:

```
label: GRAPHIC FUNCTION (argument-list);
```

The Graphic Function Header statement must have a label which is not an indexed label. It is the name by which the Function is referenced. The rules for construction of the "argument-list" are discussed in Section 2.3.5. While variables represented in the argument list may be altered during execution of the Graphic Function, such alterations do not affect the corresponding variables of the calling Routine. A Graphic Function may not invoke itself.

3.4.3.1 The MAP Statement

The syntax of the MAP statement is as follows:

```
MAP (nx,ny) [AT(r,c)] specification;
```

Within "(nx,ny)", "nx" specifies the number of columns and "ny" the number of rows of dots which are to be contained in the Point Graphic. The "AT (r,c)" enables repositioning of the cursor "r" lines from the top of the screen and "c" character positions from the left of the screen before the graphic is produced. If omitted, the current cursor position is assumed. The bottom left dot of the cursor position is the bottom-most row and the left-most column dot of the area on the screen where the graphic is to appear. The bottom left hand dot of the graphic is called the ORIGIN, and has coordinates (0,0). The bottom left hand dot of the cursor at the time of display is taken as the origin.

Indication of which of the dots are ON is provided in the "specification". The first item in "specification" must be either "HEX" or "BIN" to indicate the form of the graphic specification. For each row of dots, a binary string is required. In the string, a zero means the dot is left unchanged, and a one means the dot is to be ON. The beginning of the row description is indicated by an asterisk character. If "BIN" is specified, each row description is defined by a series of 1's (dot ON) and 0's (dot unchanged). If "HEX" is specified, each row description is defined by a series of hexadecimal characters each representing four dot indicators. (For example, 0 represents 0000, 5 represents 0101, A represents 1010 and F represents 1111.) Each row description may contain up to "nx" binary descriptors, or "nx" divided by four (rounded up) hexadecimal characters. If less than "nx" dots are specified, the description applies to the beginning of the row and the remainder of the dots are left unchanged. If more than "nx" dots are specified, the extra specifications are ignored. Spaces within a row specification are treated as the character zero.

There may be up to "ny" such row descriptors. These describe the rows of the Point Graphic in turn, beginning with the top row. There may be any number of spaces between row descriptors. If there are less than "ny" row descriptors, the remaining row dots are left unchanged. If there are more than "ny" row descriptors, these are ignored. The last row descriptor must be followed by a semi-colon.

3.4.3.2 The PLOT Statement

The PLOT statement syntax is as follows:

```
PLOT name1(argument-list1) [,name2(argument-list2)]...;
```

"name₁", "name₂"... are the names of other Graphic Functions.

The "argument-lists" are constructed according to the rules for construction of argument lists. The PLOT statement causes the specified Functions to be executed in the order (left to right) in which they appear. The PLOT statement provides the only means by which a Graphic Function may invoke another Graphic Function. Note that a position of the cursor is implied upon completion of any Graphic Function and that the cursor's position establishes the origin for any Graphic Function subsequently invoked.

3.4.3.3 System-supplied Graphic Functions

While the MAP statement permits the display of Point Graphics, there are no corresponding means of enabling the direct display of Vector Graphics. These are produced indirectly through the invocation of one or more system-supplied Graphic Functions. These are also called Elemental Graphic Functions.

ORIG (x,y) (Mnemonic for ORIGin)

Within a Function, ORIG permits over-riding of the current Origin as established by the residual cursor position. ORIG establishes a new origin at "x" dot columns right and "y" dot rows above the bottom left hand dot on the screen. The character asterisk may also appear in place of "x" or "y". This indicates that that coordinate of the new origin is to be the same as the coordinate of the current origin.

SETP(x,y) (Mnemonic for SET Pen)

SETP causes the pen to be positioned "x" dot columns right and "y" dot rows above the origin dot. Asterisk may replace either "x" or "y" to indicate that that specification remains unchanged.

MOVE (nx,ny)

This Function causes the position of the pen to be established "nx" dot columns and "ny" dot rows away from its current position. If "nx" is positive, then the new position is "nx" dot columns to the right; if negative, then "nx" dot columns to the left. Similarly, "ny" positive indicates up while "ny" negative indicates down.

JOIN (x,y [,x,y]...)

JOIN causes a straight line to be displayed starting from the current pen position to the position specified by "x,y" ("x" dot columns and "y" dot rows from the origin). The current pen position is then moved to this new position. If further sets of coordinates follow, the points identified are similarly joined in turn.

DRAW (nx, ny [,nx,ny]...)

DRAW causes a straight line to be displayed starting from the current pen position and ending at a position "nx" dot columns and "ny" dot rows from the current position. The current pen position is set at the end of the line. "Nx" and "ny" may be positive (indicating to the right or up) or negative (indicating to the left or down). Successive pairs of specifications cause further straight lines to be displayed, beginning at the end of the last line and ending "nx" columns and "ny" rows from that point.

XYXY (nx, ny [,nx,ny]...)

Starting from the current pen position, XYXY produces a straight horizontal line from the current pen position to a position "nx" columns to the right (if "nx" is positive) or left (if "nx" is negative). Then from this new pen position it produces a straight vertical line in either the upwards ("ny" positive) or downwards ("ny" negative) direction. If more "nx" and "ny"s are present, a similar action is taken for each of these; alternate horizontal and vertical lines are produced. The pen position is left at the end of the last such line drawn.

CRCL (nx, ny, degrees) (Mnemonic for CiRCLe)

This Function displays all or a part of a circle with centre "nx" columns and "ny" rows from the current pen position. The current pen position is the point at which the arc begins. The arc will be produced "degrees" degrees counterclockwise if "degrees" is positive and clockwise if negative. The current pen position is left at the end of the arc.

CHAR (character string) (Mnemonic for CHARacters)

CHAR causes the horizontal display of the specified character string starting with the current cursor position (the current pen position is at the bottom left corner of the current cursor position). "Character string" may be either a literal character string (enclosed in apostrophes) or the name of a string variable. The cursor position (and thus the current pen position) are left at a point immediately following the character string. The string may contain any of the Display sub-language control elements which are permitted in AS-IS mode. Note that normally Graphic displays overwrite material previously on the screen (i.e., they do not automatically erase). However, the area of the display accessed by the CHAR function is erased unless the appropriate &E control command is present to cause overwriting.

FILL (nx,ny)

The FILL Graphic Function will produce a rectangle within which all dots are ON. The rectangle extends "nx" dot columns right (or left if "nx" is negative) of the current pen position and "ny" dot rows above (or below if "ny" is negative) the current pen position, the pen position is left unchanged.

ERASE (function-calls)

Normally, Graphic Function calls cause selected dots to be turned ON; this is reversed by the ERASE Function which does not turn any dots ON but turns OFF all dots indicated by "function-calls". "Function-calls" is simply a series of normal references to Graphic Functions separated by commas. Any of these which normally cause dots to be turned ON have the effect of turning these same dots OFF when presented within the parentheses of the system ERASE Graphic Function.

Example

SHADE: GRAPHIC FUNCTION (NR,NC);

```

/*
 * THIS FUNCTION WILL "SHADE" A RECTANGLE DEFINED*
 * WITH LOWER LEFT CORNER AT THE IMPLICIT CURSOR *
 * POSITION AND THE OPPOSITE CORNER NR ROWS AND *
 * NC COLUMNS ABOVE AND TO THE RIGHT. *
 */
N1,N2,N3,N4 <-0;
DO I = 0 TO NR+NC BY 2;
  IF (I<NR) THEN N1 <- I;
  ELSE DO; N1 <- NR;
  N2 <- I - NR;
  END;
  IF (I<NC) THEN N4 <- I;
  ELSE DO; N4 <- NC;
  N3 <- I - NC;
  END;
  PLOT SETP(N2,N1),JOIN(N4,N3);
END;
END;

```

RHAND; GRAPHIC FUNCTION;

```

/*
 * THE FOLLOWING FUNCTIONS DRAWS A PICTURE OF A HAND *
 * POINTING ONE FINGER TO THE RIGHT. IT IS ABOUT *
 * THREE CHARACTERS LONG AND TWO HIGH. THE ACTUAL *
 * DOT MATRIX IS AS FOLLOWS (BLANK=OFF, *=ON). *
 */
*****
*
*          **          *
*        ***  ***** *
*      *  ***** *
*    ***          *
* *****          *
*          *
*        ***** *
*          *
*        ***** *
*          *
* ***** ***** *
*          *
*****
*
*
*/

```

MAP (25,15) HEX

```

*FFFFFF8*8000008*8018008
*80E7808*810FFC8*8E00028
*F003FC8*8000808*801F008
*8001008*803F008*8002008
*FEFC008*8000008*FFFFFF8;
END;

```

3.5 THE FONT BLOCK

3.5.1 Purpose of the Font Block

The Font Block provides the means whereby the user may specify a redefinition of the graphic symbols displayed as characters. Use of the Font Block is meaningful only in conjunction with devices which permit the redefinition of character graphics. Execution of a Font Block in a Display destined for an appropriate device causes an action similar to changing the typeball on a "SELECTRIC"* typewriter.

3.5.2 Invoking a Font Block

Font Blocks are invoked by the FONT keyword on a Unit Display statement, or via the &F control element in the Display sub-language. The invocation causes no action in itself, but all subsequent material presented for display is displayed using the new graphics associated with each character by the invoked Font Block.

* Registered Trademark of International Business Machines Corporation.

3.5.3 Constructing a Font Block

Each Font Block defines an entire character set. Any character which is not associated with a new graphic is, by default, associated with the standard graphic for that character. All characters available for specifying display text, except the space character, may have their graphic redefined. Graphics which may be displayed but are not associated with a character for input are assigned to &Xn elements, as are control characters such as backspace and index. These also may be redefined.

Only three types of statements may appear in a Font Block: the Header statement, the END statement, and the CHARACTER statement. There must be exactly one Header statement which is the first statement in the Block and exactly one END statement which is the last statement in the Block. Between these there must be one CHARACTER statement for each character graphic reassigned.

The Font Block Header statement is as follows:

```
label: FONT BLOCK;
```

The Header statement must have a label which is not an indexed label.

The label is the name which appears with the &F element for invocation of the Block.

3.5.3.1 The CHARACTER Statement

The CHARACTER statement may take either of two forms:

1. CHARACTER x (nx,ny) specification;
2. CHARACTER x keyword;

In both forms, "x" is the character to be redefined. It may be a single character or take the form "Xn" or "Xnn".

In Form 1, "specification" defines the character graphic by either the POINT method or the VECTOR method. The values of "nx" and "ny" identify the extent of the character being defined as "nx" dots wide and "ny" dots high. Both "nx" and "ny" must be literal positive integers.

If the specification is in Point form, it consists of "ny" row specifications. Each row specification begins with an asterisk character and is followed by a series of space and "X" characters. The character graphic is considered as a matrix of dots; each of which may be ON or OFF. The first row specification establishes the status of the dots in the first row of the matrix: "space" indicates OFF, "X" indicates ON. The rows are defined consecutively, from top to bottom by successive row specifications. There may be any number of spaces between the end of one row specification and the beginning of the next. The last row specification must be followed by a semi-colon.

If the specification is in Vector form, then it consists of a series of references to system-supplied or user-written Graphic Functions. The Function references appear one after the other separated by commas. They are invoked and executed in the order (left to right) in which they appear. The last reference must be followed by a semi-colon.

In either form, the Character specification may define a character of any size. What is actually displayed however, is determined by the area allowed for the display of characters. If the defined character graphic is larger than the permitted display area, then the bottom left hand portion of the graphic that will fit into the allowed area is displayed. If the defined character graphic is smaller than the allowed area, the graphic is displayed nearest the lower left corner of the area.

Form 2 of the CHARACTER statement permits the user to assign or reassign certain control operations to characters. The control operations that may be transmitted by characters are dependent upon the types of devices available. Keywords are assigned to the control functions. Exemplary keywords are:

BACKSPACE: Establish the next print position one character position to the left of the current print position.

INDEX: Establish the next print position on the half-line immediately below the current print position.

3.6 SYSTEM FUNCTIONS AND PSEUDO-VARIABLES

System Functions are Functions that are provided by the system. They are invoked in exactly the same way as user-defined Functions. A number of these are discussed at appropriate points in this manual; these include the Comparison Functions (Section 4.1.6), the standard Edit Functions (Section 4.1.5), and the basic Graphic Functions (Section 3.4.3). Further General Functions are defined here.

Pseudo-variables may be referenced in most cases in the same way as system variables. They differ from normal variables in that their value is determined by the system at the time they are referenced; they do not have a permanent value stored at some location.

3.6.1 Numeric System Functions

n <- ABS(x)

The absolute value function returns the absolute value of the parameter. If "x" is a null vector, a null vector is returned. If x is a single value, the absolute value is returned. If "x" is a vector of values, a vector with the absolute value taken on each element is returned.

n <- INDEX(string, item)

The Index Function scans "string" for the first occurrence of "item" and returns the integer character position where the match begins. Both "string" and "item" must be character strings. If "item" is not found, the value zero is returned.

n <- LEN(argument)

The Length Function returns an integer value equal to the length of "argument". If "argument" is a character string, it returns the number of characters; if a vector, the Function returns the number of currently defined positions; if a single switch or numeric, the value one is returned; if the variable is undefined or a Null string, zero is returned.

n <- LMATCH (string1, string2)

The LMATCH function returns the location of the first failure in an attempt to MATCH "string1" to "string2". If the two strings are identical, LMATCH returns the value zero.

n <- LOG(x)

The LOG Function returns the value of the logarithm to base ten of "x". The argument "x" must be greater than zero.

n <- LOGE(x)

The LOGE Function returns the value of the Naperian Logarithm of "x". The argument "x" must be greater than zero.

n <- MAX(argument)

The MAX Function returns the single value of the "argument" element with the greatest value. The "argument" may be a vector or pseudo vector.

n <- MIN(argument)

The MIN Function returns the value of the "argument" element with the lowest value. The "argument" may be a vector or pseudo vector.

n <- MOD(value, divisor)

The MOD Function returns the remainder left when the largest integral multiple of "divisor" has been removed from "value". Both "value" and "divisor" must be integers.

n <- NUMBR(string)

The argument "string" must be a string. The string is scanned for a character representation in the form of a literal numeric. If one is found, it is converted to a numeric and returned. If more than one are found, a vector is returned. If none are found, a Null vector is returned.

n <- NUMSTR(string, item)

NUMSTR returns an integer count of the number of occurrences of "item" in "string". Both arguments must be strings.

n <- SIN(x)

n <- COS(x)

n <- TAN(x)

The three trigonometric functions return the sine, cosine, or tangent (respectively) of "x". The argument must be a numeric expressing an angle in degrees. For the tangent of an odd multiple of 90 (degrees), the largest possible numeric is returned.

n <- PICKN(mean,sd,num)

The "mean", "sd", and "num" must be numerics. PICKN returns "num" real numeric values picked at random without replacement from a Normal distribution of mean "mean" and standard deviation "sd". If "num" is omitted, one value is returned. If "sd" is omitted, it is assumed to be one.

`n <- PICKU (lolimit,uplimit,num)`

The "lolimit", "uplimit", and "num" must have integer values. PICKU returns "num" integer values picked at random without replacement from a uniform distribution of integers between "lolimit" to "uplimit" inclusive. If "n" is the number of integers in the distribution, and if "num" is greater than "n", then only the first "n" values returned will be selected without replacement. If "num" is omitted, one integer is returned. If only one parameter is provided, it is assumed to be "uplimit" and "lolimit" is assumed to be 1.0.

`n <- PMATCH (string1, string2)`

PMATCH compares "string1" to "string2" and returns the percentage of character positions which contain the same characters in both strings. If "L" is the length of the longer string and "M" is the number of positions containing the same character, then $100 \cdot M / L$ is the value returned by PMATCH.

`n <- RND(n)`

RND rounds the value of "n" to the nearest integer. If "n" is a vector then a vector of rounded values is returned.

`n <- SQRT(x)`

The square root function returns the positive square root of the parameter if the parameter is positive, and the negative square root of the absolute value of the parameter if the parameter is negative. If the parameter is a vector then a vector of the same length is returned; each element processed as above.

n <- SUBVEC (vector, posn, len)

"Vector" may be a vector or pseudo vector, "posn" and "len" must be integer numerics. SUBVEC is intended to facilitate the recomposition of vectors. It returns that portion of "vector" beginning at element location "posn" and extending for "len" element positions. If "posn" is positive, it indicates the element position counting from the beginning of the "vector", if negative, "posn" indicates the element position counting backwards from the last element in "vector". If "len" is positive, it indicates the number of elements following the "posn" element; if negative, the number of elements preceeding the "posn"th one. If "len" indicates more elements than are available in "vector", then all remaining elements in the direction indicated by the sign of "len" is assumed. If "len" is omitted, all remaining elements following the "posn"th element is assumed. If "posn" is omitted, the value 1 is assumed. A Null vector is returned if "vector" is a Null vector, "posn" is zero or indicates an element not in "vector", or "len" is zero.

n <- TIME (time1, time2)

TIME calculates the absolute time difference between "time1" and "time2". Both "time1" and "time2" must be in the form of the system pseudo variable TIME0. Returned is a six element numeric vector containing the number of years, months, days, hours, minutes, and tenth seconds by which "time1" and "time2" differ.

n <- TRUNC(n)

TRUNC truncates the value of "n" to an integer. If "n" is a vector, then a vector of truncated values is returned.

3.6.2 Numeric Pseudo Variables

MCRSE (mnemonic for Minutes this CouRSE)

When referenced, MCRSE returns a numeric value equal to the elapsed time in minutes (to the nearest tenth second) that the student has been signed-on to the current Course.

MSESN (mnemonic for Minutes this SESSion)

When referenced, MSESN returns a numeric value equal to the elapsed time in minutes (to the nearest tenth second) that the student has been signed-on during the current session.

NCRSE

NCRSE contains an integer equal to the number of students currently executing the Course in which NCRSE is referenced.

NULL

When referenced, NULL acts as a Null label, Null string, or Null numeric or switch vector. It may appear in any position that a normal variable may appear. Values assigned to NULL by Assignment are simply ignored.

3.6.3 String System Functions

s <- SHDN (string)

The SHDN Function returns a string of the same length as "string" with every alphabetic character shifted to lower case. Non-alphabetic characters are unchanged.

s <- SHUP (string)

The SHUP Function returns a string of the same length as "string" with every alphabetic character shifted to upper case. Non-alphabetic characters are unchanged.

s <- STRING(n, precision, chars)

The STRING Function will convert a numeric "n" into a character string of length "chars". "Precision" places of decimal are provided. The result is right-justified in the field and preceded by spaces if necessary. If the numeric cannot be expressed in character form as requested, a string of asterisks results. The "n" may be a vector or pseudo vector; the string produced is then a concatenation of the character representations of the values in the order in which they appeared in the vector. If "chars" is omitted, the length of the string will be just sufficient to satisfy the "precision" requirement. If both "chars" and "precision" are defaulted, normal numeric to string conversion as discussed in Section 2.3.3 is indicated and the character representations are separated from each other by a space character.

3.6.4 Pseudo Functions

Two items are defined which behave partly like Functions and partly like Pseudo Variables. They may appear on either the left or the right hand side of an assignment statement.

s <- SUBSTR(string, posn, len)

"String" must be a character string, "posn" and "len" must be integer numerics. In this form of use of SUBSTR, the result "s" is set to that part of "string" beginning at character position "posn" and continuing for "len" characters. The Null string is returned if "string" is Null, "posn" is zero or greater in absolute value than the length of "string", or "len" is zero. If "posn" is positive, its value indicates the character position counting from the beginning of the string; if negative, counting from the end of the string. If "len" is positive, it indicates the number of characters to the right of "posn"; if negative, to the left. If "len" indicates more characters than are available, then the remaining characters are indicated (in the direction determined by the sign of "len"). If "len" is omitted, the remaining characters to the right ("posn" positive) or left ("posn" negative) is assumed. For example, assume W contained "ABCDEF", then the following would hold:

SUBSTR(W,4,1)	would yield	'D'
SUBSTR(W,-3,1)	would yield	'D'
SUBSTR(W,2)	would yield	'BCDEF'
SUBSTR(W,-2)	would yield	'ABCDE'

SUBSTR (string,posn, len) <-s

In this form of use of SUBSTR, "string" is modified. A region of "string" is indicated by "posn" and "len" in the same way as discussed with the first form of use of SUBSTR. This region is replaced by "s". If "s" is Null, "posn" is zero or greater than the length of "string", or "len" is zero, then "string" is not changed.

s <- WORD (string, posn, len)

Spaces, digits, and symbols are considered to delimit items called words by the WORD Pseudo Function. A word consists of a string of alphabetic, digits, or symbols with no intervening spaces. A word boundary is determined by every space and every shift between digits, symbols, and alphabetic.

"String" must be a character string, and "posn" and "len" must be numerics. In this form, WORD returns a portion of "string" identified by "posn" and "len". The "posn" indicates a starting word of the portion to be returned, and "len" indicates the returned portion's length. If "posn" is positive, its value indicates the word counting from the beginning of the "string"; if negative, counting from the end of the string. If "len" is positive, it indicates the number of words to the right of the identified word, if negative, to the left. If "len" indicates more words than are available, the remaining words (in the direction indicated by the sign of "len") is indicated. If "len" is omitted, the remaining words to the right ("posn" positive) or left ("posn" negative) are assumed. The Null string is returned if "string" is Null, "posn" is zero or indicates a word not in "string", or "len" is zero.

Example, suppose W contained "one, two, 3/4"

There are seven words in this string: "one", ",", "two", ",", "3", "/", and "4".

```
WORD(W,3,1) yields "two".
WORD(W,-5,1)yields "two".
WORD(W,2)   yields ", two, 3/4"
WORD(W,-2)  yields "one, two, 3/"
```

WORD (string, posn, len) <-s

In this form of use of WORD, "string" is modified. A region of "string" is indicated by the numerics "posn" and "len" in the same way as discussed with the first form of use of WORD. This region is replaced by "s". If "s" is Null, "posn" is zero or greater than the length of "string", or "len" is zero, then "string" is not changed.

3.6.5 String Pseudo Variables

TIME0 This variable is a fixed length character string (20 characters) in the form "YEAR-MO-DD HH MM SSS" where "YEAR" is the four character year, "MO" is the two character integer number of the month,, and "DD" is the two character integer number of the day; HH is the two-digit hour, MM is the two-digit minute, and SSS the three-digit tenth second of the time at which the particular student for whom the pseudo-variable is referenced began the course. The space and hyphen characters appear as shown.

TIME1 A variable of the same type and format as TIME0 but containing the date and time that the student began the current session.

TIME2 A variable of the same type and format as TIME0 but containing the current date and the time at which the variable was referenced.

Within the language, the direct interface between the student and the actions of the computer occurs in the structural component called the Instructional Unit (or simply Unit). Conceptually, each Unit is intended, and should be constructed to encompass one complete and self-contained instructional transaction between the computer and the student. This transaction may: provide the student with necessary information, ask a question concerning the topic at hand, examine and categorize the forthcoming response and finally either provide a reinforcement message or clarify the question and request another response. These components of the transaction are facilitated and encouraged by the structure of the Instructional Unit.

Units are invoked only by Procedures, and have access to a built-in sub-language for the display of material to students and to the Service Functions of the previous section. In this section, the statements permitted within Units are presented and the means of constructing Units is discussed.

4.1 INSTRUCTIONAL UNIT STATEMENTS

Seven types of statements are permitted in an Instructional Unit or Unit. Two of these, the Unit Header statement and the END statement must be present; the remainder are optional. Labels on statements other than the Header statement are ignored.

4.1.1 The Unit Header Statement

`*unit-label : UNIT [(argument-list)];`

The "unit-label" consists of an alphanumeric character string. The first character must be an asterisk, the second an alphabetic. The remainder may be alphabets or digits or the underscore symbol. The "unit-label" may be an indexed label by appending "#n", where "n" is a literal integer greater than zero, to the label. For example, "UNIT#2".

Once entered into the system, the Unit is "known" by this label. Every Unit must have an indexed or non-indexed label; no two Units may have the same label.

The text "UNIT" identifies the statement as a Unit Header statement. It is optionally followed by an "argument-list" enclosed in parentheses. All variables except system variables referenced in the Unit must be identified in the argument list.

The Unit Header statement signals the beginning of a Unit, provides the Unit's label, and identifies all input variables required in the Unit. Each Unit has exactly one Header statement as the first statement of the Unit.

4.1.2 The END statement

END;

The text 'END' identifies the statement as the END statement. Each UNIT has exactly one END statement as the last statement in the UNIT. The END statement indicates both the physical and logical end of the Unit.

4.1.3 The DISPLAY Statement

DISPLAY [ON device] text

The text 'DISPLAY' identifies the DISPLAY statement. The DISPLAY statement identifies a request for display of "text" on a device. The device may be determined by default or specified if alternative choices are available at the particular installation. A logical name associated with a particular class of devices is used in this latter case. Examples are "PRINTER" and "SCREEN". If a logical device name is present, but this name does not correspond to any logical device name available to the particular student executing the Unit, the ERROR condition is raised and the default device is used. The "device" addressed by a DISPLAY statement must be capable of accepting text. Other types of devices (slide projectors, audio units) are controlled by the &D control command element.

"Text" consists of an alphanumeric character string which begins with any Display sub-language command and ends with a special Display sub-language control command. The rules for determining the "text" portion of the DISPLAY command is discussed under the Display Sub-language in Section 4.3.

4.1.4 The RESPONSE Statement

RESPONSE [ON device] qualifier-list;

The text "RESPONSE" identifies the RESPONSE statement. This statement indicates a request to accept input from a device. The device may be determined by default or specified by "ON device" where "device" is the logical name of an available device (for example "KEYBOARD"). If a device is specified which is not available to the person executing the statement, the ERROR condition is raised and the default device is used. The received response is placed in both the RESPONSE and REDITED system variables. This consists of either the character text of a keyboard response or the character representation of a graphic "point" response in the form row index, comma, column index.

The "qualifier-list" is a list of qualifiers separated by commas.

Qualifiers may consist of a single word or take the form "keyword=quantity".

The qualifier syntax and semantics are as follows:

CHARS=n: The Character Limit qualifier indicates that the student will not be allowed to enter more than "n" characters or graphic "points" in his

response. "N" must be an integer expression greater than zero. If the device permits, entry of more than "n" characters is inhibited. In any case, only the first "n" characters are placed in or appended to the RESPONSE buffer. If CHARS is not specified, the default maximum of 100 characters or a single graphic response is established.

TIME=n The Time Limit qualifier indicates the maximum number of seconds during which the student must complete his response. "N" must be a real value expression greater than zero. If the device permits, further input is inhibited once the Time Limit has expired. If no time limit is set, a default limit indicated by STIME is used. If a student does not complete his response within the required time, he is said to have "timed-out" and the FOTIME flag is set, the NOTIME counter is incremented and the OTIME Condition is raised.

POSN=(r,c) The Position qualifier specifies the location of the first character position of where the students response appears as he types it. POSN is meaningful only for keyboard responses. Both "r" and "c" must be positive integer expressions which identify the number of the row (counting from the top) and the number of the column (counting from the left) respectively. If the device does not have random display access, POSN is ignored and the response appears in the next available row. If either "r" or "c" is zero or larger than the number of rows or columns (respectively) available on the device, then the response is not displayed as it is typed.

TYPE=type The Type qualifier identifies the type of response permitted. The "type" is a one character alphabetic. TYPE=A indicates that a keyboard response is permitted, TYPE=B indicates that either a keyboard or graphic response is permitted, and TYPE=C indicates that only a graphic response is permitted. The meaning and use of the remaining alphabetic types is reserved for future expression. If a response type is requested which is not available to the student, the default type (TYPE=A) is used. If the received response is not of the specified type, the RTYPE condition is raised.

FONT=name The Font qualifier identifies the particular character set to be used for display of the student response as he types it. If the qualifier is omitted, the graphics of the default Font Block are used.

APPEND At the execution of a RESPONSE statement, the received input normally replaces the contents of the system variable RESPONSE. If the APPEND keyword is present on the RESPONSE statement, and if a response is requested more than once during execution of the unit, the responses are accumulated in the RESPONSE variable in the order in which they are received, with a single intervening space character. At the first execution of the RESPONSE statement after entry or re-entry to the Unit, the incoming response replaces the contents of the RESPONSE and REDITED variables whether APPEND is present or not.

4.1.5 The EDIT Statement

EDIT edit-list;

The text "EDIT" identifies an EDIT statement. At the time a RESPONSE statement is executed, the incoming student response is placed in or appended to both the RESPONSE and REDITED system variables. The entire response in REDITED is then processed by the Edit Functions named in "edit-list". The EDIT statement has no effect on the RESPONSE variable.

Two types of "edit-list" items are permitted, Edit Function names and Character-swap strings. The "edit-list" consists of a list of these items separated by commas. Editing is done by executing the named Edit Functions and performing the actions of the Character-swap Strings sequentially, from left to right, as they appear in the "edit-list". System provided Edit Functions alter the REDITED system variable as follows.

ALPHA: The ALPHA Edit Function changes all recognized literal numerics (which may contain a decimal character, a sign, or be in "E" notation) to spaces and then performs the MULSP actions.

MULSP: The MULSP Edit Function removes all leading and trailing space characters and changes all occurrences of multiple spaces to single spaces in REDITED.

NUMBR: The NUMBR Edit Function changes all non-digit characters to spaces (except: period and character "E" which are left unchanged if preceded and/or followed by a numeric character, "0" is changed to zero, and character "1" is changed to character one); then the MULSP actions are performed.

PUNC: The PUNC Edit Function changes the apostrophe, quote, parentheses, comma, semi-colon, colon, period, question and exclamation mark characters to space characters.

SHDN: The SHDN Edit Function shifts every upper case alphabetic character to its corresponding lower case equivalent.

SHUP: The SHUP Edit Function shifts every lower case alphabetic character to its corresponding upper case equivalent.

STANDARD: The STANDARD Edit Function removes all codes from the response which do not represent characters in the Extended Character set. Examples of such characters are the tabulation character, index or line-feed character, return character, etc. It also performs the MULSP operation.

SYMB: The SYMB Edit Function changes every character which is not an alphabetic character to a space; then the MULSP actions are performed.

CHARACTER SWAP STRINGS: The second type of edit item, the Character-swap string, uses a delimiter which may be any of the symbols except slash and semi-colon. The first symbol of the Character-swap string identifies the delimiter and the same delimiter must be used as required within that edit-item. For purposes of explanation the apostrophe will be used for the delimiter.

The Character-swap string takes the form:

'string 1'string 2'

It indicates that all occurrences of "string 1" in REDITED are to be replaced by "string 2". "String 2" may be a null string to cause the removal of all occurrences of "string 1" from variable.

The user may define his own further edit-list items by means of the Edit Functions. The appearance of an Edit Function name in the edit-list causes that Function to be invoked. Note that parameters cannot be passed to Edit Functions.

4.1.6 The Categorization Statement

category target;

There are sixteen category types: F0, F1, F2, F3, F4, F5, F6, F7, F8, F9, OTIME, RENTRY, RIGHT, RTYPE, UNREC, and WRONG. Any one of twelve of these (F0, F1, ... F9, RIGHT, or WRONG) may appear in the "category" portion of the Categorization Statement. The "target" is composed of a single-valued expression, usually containing system supplied or user written comparison functions. If the result of evaluation of "target" when converted to a logical value is TRUE, then the Category named by "category" occurs, and a Condition of the same name is raised. Occurrence of a Category causes a corresponding system switch F1, F2...F9, FRIGHT, or FWRONG to be set and a system numeric NF1, NF2, ... NF9, NRIGHT, or NWRONG to be incremented by one. If no category is raised in the Unit, the UNREC Condition is raised, the UNREC Category occurs, FUNREC is set, and NUNREC is incremented.

4.1.6.1 System-Supplied Comparison Functions

The Categorization statement target is a logical expression which may make use of logical functions called Comparison Functions. Six Comparison Functions are supplied but more may be devised by authors.

The Alphameric Comparison Function

CC(target-string [,word][,spec])

The "target-string" may be a literal character string (enclosed in apostrophes) or a string expression. The optional second parameter "word" is a numeric non-vector expression. The purpose of this Function is to attempt a character match between the response and the "target-string". The "target-string" may contain two special characters, "*" and "&". The "*" character will match any single character in the response; and the "&" character will match any string (including the Null string) of characters in the response. The "target-string" may also contain parenthesis for the purpose of grouping alternative responses; commas must be used to separate the alternatives. For example, if the "target-string" were:

'(ALTA, ALBERTA) AND (B.C., BRITISH COLUMBIA)'

then the following responses would match:

- ALTA AND B.C.
- ALTA AND BRITISH COLUMBIA
- ALBERTA AND B.C.
- ALBERTA AND BRITISH COLUMBIA

If "word" is included, then comparison between the target and the response begins at that word in the response. (A word here is a string of non-space characters delimited on either side by space characters or the end of the string.) If omitted or non-positive, "word" defaults to 1. If "spec" is included, it may be either a Null string or a string of five characters. If it is a Null string then the effect of the special "&", "*", "(", ",", and ")" characters is inhibited and they simply represent themselves. If not Null, the first two characters of "spec" are used to indicate the action requested by "*" and "&" respectively, and the remaining three the action requested by "(", ",", and ")" respectively.

The Numeric Comparison Function

CN(numeric-target [,tolerance[,word]])

The three parameters must all be numeric expressions. The purpose of the numeric comparison function is to attempt to match the "numeric target" with a numeric response given by the student. The "tolerance" is the allowable absolute difference between the target and response. The "word" indicates which number in the response is to be compared to the numeric target. (Here, word is a string of digit characters which may contain a decimal or "E" character delimited on either side by non-numeric characters or the end of the string.) If "tolerance" is omitted, the response must be numerically equivalent within the limits of error of the computer for a match to occur. If "word" is omitted, the first word is assumed. The NUMBR Edit Function is applied to the current response being tested before comparison is made; this has no effect on subsequent comparisons.

The Phonetic Comparison Function

CP(phonetic-target [,word])

The Phonetic Comparison Function attempts a phonetic comparison between "Phonetic-target" and the student's response, beginning at the "word"th word of the response. (Here, word is a string of alphabetic characters delimited on either side by non-alphabets or the end of the string.) Its purpose is to correctly recognize responses which may contain spelling errors. If an exact match between the target and the response fails, a comparison is made using the method described by Symonds (1970)*. The method first checks for an error resulting from a single instance of substitution, deletion, or insertion of a letter, or transposition of two adjacent letters. If a match is not found, the phonetic test is conducted in the following manner. For each response word, the function produces a set of "pronunciations" based on the occurrence and order of consonants and some common digrams and trigrams in the word. These are compared to similarly produced pronunciations of the corresponding target word, and a word match is indicated if any response word "pronunciation" is equal to one from the corresponding target word.

The number of possible pronunciations of a word depends upon the letters appearing in the word and their common pronunciations, and the language used. The CP Function assumes that the applicable language is that specified in the system variable SLANG. However, similar Functions named CPE and CPF are available which assume the languages ENGLISH and FRENCH respectively.

*

Symonds, M., Computer Detection of Misspelled Words, National Research Council of Canada, Ottawa, 1970.

If the "phonetic-target" contains more than one word, each word must phonetically match the corresponding response word for an overall match to occur. Symbols other than alphabets must not appear in the "phonetic-target"; non-alphabets are treated as word delimiters in the response. No distinction is made between upper and lower case letters in execution of the Phonetic Comparison Function.

The Algebraic Comparison Function

CA(target,vec1 [,vec2]...)

The Algebraic Comparison Function is provided to compare the Student's response to a "target" algebraic expression. The arguments "vec1", "vec2", ... are vectors. There must be one vector argument for each variable in the "target" expression. A match occurs if all of the following conditions are met.

1. The response must contain the name of every variable in the target expression at least once, and may not contain the names of any variables not in the target expression.
2. The response must contain only variable names, numeric literals, arithmetic and relational operators, parentheses, and spaces (as must the target).
3. Values are associated with each variable; using these the response and the target, when evaluated, must have the same value within the limits of error of the computer. A selection of values and an evaluation occurs once for every different variable name present in "target". The match fails if any of the evaluations produce different results. If any evaluation of the target results in an undefined value (e.g. division by zero), the ERROR Condition is raised.

The values selected for substitution for each variable are determined by the vector arguments. There must be the same number of vector arguments as there are different variable names in the target. The "vec1" is associated with the left-most variable name in "target", "vec2" is associated with the next different variable name, and so on. Each vector may be Null, or have one, two, or N elements where N is the number of different variable names in "target". If a vector is Null, then a random number is chosen for substitution with the associated variable for each evaluation. If it has one element, then the value of that element is substituted for the associated variable for all evaluations. If it has two elements, these are taken as the lower and upper limits (respectively) on the random numbers chosen for substitution. If the vector contains N elements, these N values are used, in turn, as substitutes during the N evaluations.

The Graphic Comparison Function

CG(r,c,nr,nc,word)

The five parameters must all have integer values. The purpose of the Graphic Comparison Function is to determine whether a specific student graphic (point) response was within a specified graphical region. The target region is defined by "r" the top-most row, "c" the left-most column, "nr" the number of rows, and "nc" the number of columns. "Word" allows the selection of a single graphic response of more than one are in the response buffer. If "word" is defaulted then a match occurs if any of the responses are within the required region. "r" and "c" default to one and "nr" and "nc" default to the number of remaining rows or columns respectively.

The Keyletter Comparison Function

CK (target-expression, criterion)

The "target-expression" may be either a literal character string (enclosed in apostrophes) or a string expression; "criterion" must be a numeric expression. The "target-expression" is compared to the student response as follows. The student response is scanned for a word (a sequence of characters which contains no spaces) containing all of the characters of the first word in the "target-expression" in the same order. The student response word (but not the target word) may contain additional characters (except spaces) interspersed between the matching characters. If a word-match is found, the response is scanned for word-matches on the second and subsequent words. The matching words must be in the same order in the response as they are in the "target-expression". A count of matching words is then made and if this is greater than or equal to the value of "criterion", an overall match occurs. "Criterion" may be omitted indicating that all target words must be found in the response.

User Defined Functions

The user may define further Comparison Functions of his own. These are simply General Functions which return a logical result.

4.1.6.2 Comparison Target Assembly

Each of the Comparison functions results in a logical value of true or false. These functions may be combined according to the rules of logical

operations to produce logical expressions. The target portion of the Categorization statement consists of this logical expression. The target is evaluated and found to be true or false and the corresponding condition is raised or not raised accordingly.

All of the Categorization statements in a Unit may be executed so that it is possible for up to twelve simultaneous conditions to be raised. It is therefore important to consider the order of appearance of the Categorization statements and the Reprise statements in the Unit. This topic is discussed under Order of Execution.

4.1.7 The Reprise Statement

The Reprise statement may take either of two forms:

RETRY category text

REINF category text

A reprise statement is identified by the text 'RETRY' or 'REINF'. Its purpose is to take some immediate action as a result of one of the categories set by the Categorization statements. The "category" identifies the particular category to which the statement applies. "Category" may be one of RIGHT, WRONG, F0, F1, ... F9, OTIME, UNREC, RTYPE, or RENTRY. "text" is the same as 'text' on the DISPLAY statement; and is discussed under the Display Sub-language.

If the Reprise statement is a RETRY statement and if the named category occurred, then the "text" portion of the statement is transmitted to the student and the Unit is re-entered at the point of the RESPONSE statement. If there is no RESPONSE statement in the Unit, then the statement execution is identical to the REINF statement execution.

If the Reprise statement is a REINF statement and if the named category occurred, then the "text" portion of the statement is transmitted to the student, and control passes back to the Procedure which invoked the Unit.

The RIGHT, WRONG, F0, F1, ... F9, and UNREC categories are explained in the discussion of the Categorization statements. The RTYPE category occurs if the student response type did not match that specified on the response statement (for example, if the student responded with a typed response when a light-pen response was specified). The OTIME category occurs if the student does not complete his response in the time limit determined by the RESPONSE statement.

Units may be invoked by both the CALL and REPEAT statements. When invoked by a REPEAT, the Reprise statements are scanned for one or more containing the text 'RENTY' in the "category" position. If found, execution begins at the next unexecuted or the last Reprise RENTRY statement. If none are found, execution commences from the beginning of the Unit.

4.2 UNIT CONSTRUCTION

4.2.1 Order of Execution

The order of appearance of UNIT statements must be as follows:

1. Unit Header statement (mandatory)
2. DISPLAY statements(optional)
3. RESPONSE statement (optional)
4. EDIT, Category, and Reprise statements (optional)
5. END statement (mandatory)

For normal execution of a Unit, the statements are executed sequentially as they appear. If invoked by a REPEAT statement, execution begins at the last unexecuted Reprise statement addressing the RENTRY category. If no such Reprise statement is available in the Unit, then execution commences with the first executable statement in the Unit. The Reprise statement is the only statement which may affect order of execution. It has an effect only if the Category which it addresses occurred.

When the interpreter first reaches the EDIT, Category, and Reprise statements, they are examined in turn: if an EDIT statement, the requested editing is performed; if a Categorization statement, then the category named occurs or does not occur; if a Reprise statement, then action is taken depending on whether or not the category is raised at that time. When a Reprise RETRY statement is executed, then the categories previously raised are cleared and interpretation of statements restarts at the RESPONSE statement. On the second and subsequent passes through the Edit, Categorization and Reprise statements, a Reprise statement is executed if the category has occurred and

only if either the statement has not been executed previously or it is the last Reprise statement addressing the particular condition.

Consider the following example:

```

1.  RESPONSE;
2.  EDIT SHUP;
3.  RIGHT ...
4.  REINF RIGHT &L Good!&;
5.  WRONG ...
6.  RETRY WRONG &L Close, but not quite &;
7.  REINF WRONG &L the answer is ... &;
8.  RETRY UNREC &L I didn't recognize that answer ... &;
9.  END;

```

The statements are numbered for reference purposes. If the student entered the "right" answer, the RIGHT category would be raised by statement 3, then statement 4 would be executed and control would return to the calling Procedure. Note that if the order of statements 3 and 4 were reversed, that the Reprise REINF statement would not be executed; the student's answer would be judged "right" but since no following Reprise statement is available, control would return to the calling procedure without a reinforcing message.

Suppose instead that the student entered a "wrong" answer. It is categorized as WRONG by statement 5; the message "Close, but not quite" is transmitted by statement 6 and execution continues with statement 1. If he enters a "wrong" answer again, it is again identified by statement 5, this time statement 6 is skipped and statement 7 is executed telling him the answer. Since statement 7 is a REINF statement, control returns to the calling Procedure.

Thirdly, suppose the student enters a response which does not raise either the RIGHT or WRONG condition. When execution reaches statement 8, the UNREC

category automatically occurs and statement 8 is executed. If the student enters any number of successive "unrecognized" responses, statement 8 is executed repeatedly as it is the last reprise statement with the UNREC condition.

Note that statement 5 could appear in any position between statements 2 and 6 without changing the order of execution. Similarly, 8 could be in any position as long as it followed statements 3 and 5; 4 could be anywhere after statement 3; and 5 and 7 could be anywhere after statement 5. An equivalent arrangement of the statements follows:

1. RESPONSE;
2. EDIT SHUP;
3. RIGHT ...
4. WRONG ...
5. REINF RIGHT &L Good! &;
6. RETRY WRONG &L Close, but not quite &;
7. REINF WRONG &L The answer is ... &;
8. RETRY UNREC &L I didn't recognize ... &;
9. END;

The fact that certain Reprise statements have already been executed is "remembered" by the system until: the calling Procedure is completed, the student signs off, or the Unit is re-invoked by a CALL *unit-name. Therefore a number of Reprise RENTRY statements may be included which are executed upon re-entry in turn until the last such statement. The last Reprise RENTRY is executed for all subsequent re-entries. Note that the Reprise REINF RENTRY statement, while acceptable, results in transmission of the text of the statement to the student and nothing else.

4.2.2 Unit System Variables

A Unit has read-only access to those variables in the calling Procedure passed in through the argument list, read and implicit write access to a number of system variables, and unrestricted access to literals. It may invoke Edit Functions, General Functions, Graphic Functions and Font Blocks. The only means by which a Unit may "return" information to its calling Procedure is through modification of system variables; this modification is done automatically while the Unit is being executed and is not otherwise controllable by the author. The Unit may raise Conditions which may be intercepted and cause some action by Condition Blocks, but these actions are transparent to the execution of a Unit and to its calling Procedure.

The system variables that are directly affected by Unit execution are as follows:

STRING VARIABLES: RESPONSE by the RESPONSE statement;
REDITED by the RESPONSE and EDIT statements.

SWITCH VARIABLES: FRIGHT, FWRONG, FO, F1, ..., F9, FOTIME,
FUNREC by Conditions raised in the Unit.

NUMERIC VARIABLES: NRIGHT, NWRONG, NFO, NF1, ..., NF9, NOTIME
NUNREC by Conditions raised in the Unit;
NRESP by the number of responses entered; ITYPE by the type of the
the student's response; and LATENCY by the latency time of the response.

4.3 THE DISPLAY SUB-LANGUAGE

Presentation of textual and graphical material to students is done through the Unit DISPLAY statement. The author controls how this material is formatted on the students' display device by means of the Display Sub-language. With the exception of the special AS-IS mode, the data accepted for transmission is a continuous stream of characters - textual material interspersed as necessary with Display sub-language control commands.

Every control command begins with the control command character and ends with any character not legal in a control command (usually a space). By default, the control command character is the ampersand '&', but this may be changed to any of the special symbols by the control command character. The control commands are not a part of the text transmitted to the student; the entire command including a trailing space, if present, is removed from the text.

4.3.1 The Control Command Elements

Control commands are constructed from the Control Command Elements, presented below. Where the modifier "(r,c)" is used, "r" indicates the rth line down from the top of the current page, and "c" the cth character position in from the left edge of the page. The first character of the first line has coordinates (1,1).

&A: Begin AS-IS Mode.

Under certain conditions, the author will want to override the automatic formatting of text. The &A control element indicates that the following textual data is to appear on the display as-it-is on the following lines. Within AS-IS mode, only the following control elements are recognized: &D, &F, &H, &Q, &U, &V, &W, &X, &Z. AS-IS mode is terminated by the control element &Z. &A further establishes single-spacing, turns off justification and cancels all previous &H, &I, &M, &O and/or &U element actions.

&B(r,c,nr,nc): Define BOX for Display

With the Box element, the author can define an area of the display screen in which material is to appear. "R" identifies the first row of the box, "c" the first column; "nr" and "nc" the number of rows and columns in the box. The element &B alone identifies the entire screen is the box (this is the default or initial setting). Thinking of the display area as divided into 4 quadrants numbered counter-clockwise from the top right as 1,2,3,4; then &B1,&B2,&B3,&B4 will define one of these quadrants as the area for display. Two or three quadrants may also be specified with this alternate notation as, for example B12 for quadrants 1 and 2 (the top half of the screen) or B23 (the left half of the screen). Once a box is defined, it remains in effect until another box is defined (or an &R control element is encountered). The defining of a box causes (unless otherwise specified) subsequent text to appear within the box beginning at the upper left corner character position. The "r" and "c" parameters of &B are ignored for serial printer (e.g. typewriter) devices. "Nr" and "nc" may not define a box that "overflows" the screen. If such is the case, "nr" and/or

"nc" are reduced to their maximum allowable values. If either "r" or "c" are not "on" the screen, they are set to their default value of one.

&C(r,c): Coordinates of Start of Text

The &C element permits the author to specify the next-text-character position where following text is to begin. Previous text (if any) is transmitted the same as if an &P were issued; if the coordinates specified fall on text transmitted within the current Box, an &WO is issued automatically. The parameters "r,c" specify the line number (from the top of the "page" down) and the character position (from the left edge of the "page") respectively.

&D(device,action,data): Device Activation

The Device element controls a device such as a slide projector or audio unit. The first parameter identifies the device type; for example, SLIDE, AUDIO, DEVICE1, etc. The second identifies the action to be performed by the device: SEEK - find a particular slide or tape section, ONW - turn the device ON and wait for a completion signal before continuing with text processing, ONC - turn the device ON and continue immediately with text processing, OFF - turn the device off. The third parameter passes digital information such as the location for a SEEK action; this parameter may be a literal vector or a variable name.

&E(r,c,nr,nc): Erase box

The &E element erases the screen area defined by "r" top-most row, "c" left-most column, "nr" number of rows, and "nc" number of columns. &E alone erases the currently defined Box. The next-text-character position is set to "r,c" unless this is outside the current Box. If "r" is outside the Box, it is set to the top-most row of the Box; if "c" is outside, it is set to the left-most column. For any CRT-type device, the material on the screen is erased before the new material is displayed. This can be suppressed by the specification of &E9. Subsequently, material is displayed without erasure of previous material. The effect of &E9 is terminated by subsequent encounter of an &E,&N, or &R element.

&F(name): Function Reference

The Function reference may invoke either Font Blocks or General Functions. The character set used to specify programs is defined, but the displayed character graphics which correspond to the available input characters may be altered. This is done by defining the new graphics in a Font Block and subsequently invoking the new set by calling the Block with the &F element. Once issued, a change of Font remains in effect until another change is requested. The standard character set is the default for each device and is the initial setting. It may be restored with an &F and no following "(name)".

In the case where "name" is a General Function reference, the element has the form "&F(name(argument-list))" General Functions always have

an explicit result and this result is inserted into the text stream in character form. If the result is logical, a character one or character zero is inserted; if a numeric, the character representation is presented if a numeric vector, the values are presented with a single intervening space; if a string, the character string is presented.

&G(name(arg-list)...): Graphic Function Call

The use of the Graphic element invokes one or more Graphic Functions which cause the display of graphic material. &G is discussed with the Graphic Function.

&Hx: Highlighting

The &H element causes subsequent text to be transmitted with extra emphasis of some form. &H with no modifier indicates normal text transmittal. The "x" modifier must be constructed of two parts; the first a single letter and the second a valid literal numeric (which may not be expressed with the "E" notation). The letter may be "A" to indicate that following text is to be transmitted with the size of each character proportional to (multiplied by) the value of the numeric. For example, &HA4 indicates following text is to be displayed four times its normal size; &H.5 indicates half normal size. The letter may be "B" to indicate the brightness or intensity of the text displayed; &HB2 would indicate double intensity. The letter "C" is reserved to indicate colour specification, and the letter "D" indicates the speed of a flashing or "strobe" effect; for example, &HD5 indicates 5 "flashes" per second. Other letter modifiers of the &H element are reserved for

future expansion. In all cases, if the letter is present but a following numeric is not present, the indicated aspect of highlighting is returned to its normal state: i.e., normal size, normal brightness, normal colour, and no flashing.

&I(l,r): Margin (Indent) Settings

For each distinct device type, three indent or margin settings are established from both the left and the right sides of the display area. "l" and "r" must be integers (0,1, or 2) to indicate which of the margins are to be used. The default and initial setting is &I(0,0), or the full width of the display area. &I alone resets this default. The form &I0, &I1 or &I2 may be used to set the left margin only (leaving the right margin unchanged). In the case of interaction between Box and Margin elements, the intersection of the two regions is used. In discussions in other parts of this section, the word BOX is used to refer to this intersection.

&J: Justification of Text

In the transmittal of textual material to a display device, the Display Sub-language Interpreter constructs the lines by fitting as many words as possible into each line to be sent. If Justification is off, the words are sent separated from each other by a single space (causing the right-hand edge of the display to vary as in this paragraph). If Justification is on however, spaces are inserted between words so that the last non-space character of each line always falls in the last permissible character position for the currently defined BOX. This occurs except when the text is followed by a control element other than &D, &F, &H, &Q, &U, &V, &W or &X.

If one or more of the other control elements are encountered, the text remaining to be transmitted is justified if it contains enough characters to fill 90% of the current line. Otherwise it is transmitted unjustified. &J switches between justify-on and justify-off. Initial and default setting is justify-off.

&Kn: Keep Text on Same Page

The Keep element causes the interpreter to compare "n" with the number of lines remaining on the current page. If "n" is larger, then an &N is issued. Otherwise the element has no effect. Note that "n" refers to single lines and is not affected by the current &S setting.

&Ln Skip n Lines

In the form &L, "spacing" (as set by &S) line(s) are skipped (and erased). Otherwise &Ln causes the skipping (and erasing) of "n" (single-space) lines. Erasing occurs unless &E9 has been specified.

&M: Center Text (Middle)

The &M control element temporarily overrides the Justify switch (if on) and causes subsequent line(s) to be centered in the currently defined BOX. The effect of &M is cancelled by a subsequent &M.

&N: New Page (or Screen)

The &N causes the automatic issuance of an &W0: then, if a printer, it advances to a new page; if a serial printer, several lines are skipped; and if a random access display device, the current screen is erased and further text begins at the upper left character position of the current Box. &N cancels the effect of any outstanding &H, &M, &O, and/or &U.

&O(l,r): Offset (Hanging) Indent

The Offset control element causes the next line to be transmitted using the current left margin and a right margin defined by 'r'. Subsequent lines have margins defined by 'l' and 'r'. (Margins are discussed under &I).

&P: Begin a Paragraph

&P causes "spacing" (as set by &S) lines to be skipped and the next line to be indented a number of characters (as determined by the installation). &P cancels any outstanding &H, &M and/or &U if in effect.

&Qa: Change Control Character

For various reasons, it may be desirable to use a character other than '&' as the Display Sub-language control character. The character is changed by &Qa where 'a' is any printable character.

&R: Reset Format Control

&R causes the following actions:

- Box is defined as the entire display area
- Font is set to the default
- Highlighting is turned off
- Indents revert to the entire display width
- Justification is turned off
- Centering of text is turned off
- Offset is cancelled
- Underlining is turned off

&Sx: Spacing Setting of Text

The &S control element controls the spacing of text. The "x" modifier may have a fractional value. For &S1, single spacing (as on a typewriter) is indicated. This is the default. For &S1.5, a half-line space appears between lines of text. Any positive value of "x" is permitted. If "x" has a fractional value and the device does not permit other than whole line access, "x" is rounded up to the next integer value.

&Tn: Tabulation

The &T element specifies the character position within the current line at which the next word is to end. The "n" may range from 1 to the width of the current display area. &T10 indicates that the last character of the next word is to appear 10 character positions to the right of the current left margin. If the specified character position has already been passed in construction of the current line, the next line is used. If the specified position has not been passed, but the placed word would overlap on a word already in the line or will not fit between the current left margin and the specified position, then "n" is incremented to the smallest higher value which will allow the word to be displayed. If "n" is or becomes larger than the current display width, only that part of the word which will fit in the current display area is printed.

&U: Underlining of Text

The &U element causes all subsequent text, except for spaces, to be underscored. Underlining is stopped by the subsequent occurrence of a &U element.

&V(name(w,d)): Variable Insertion

Variables from the calling procedure may be inserted into display material by means of the &V element. 'name' identifies the variable to be inserted by its name (as referenced in the Unit Header statement), and 'w' defines the maximum width of the field to be allowed for the variable. If the field width is too small, asterisks are presented; if too large, the variable is right justified in the field (the remainder of the field is filled with spaces).

For switch variables, 'd' indicates the form of presentation. The SET variable SLANG is examined, and if it contains 'ENGLISH', then the following forms are used. If d=1, '1' or '0' is presented; if d=2, 'YES' or 'NO' is presented; if d=3, 'ON' or 'OFF' is presented; and if d=4, 'TRUE' or 'FALSE' is presented. If a numeric variable, 'd' indicates the number of places of decimal desired. If 'd' is zero, the variable is truncated and presented as an integer; if 'd' is positive, normal decimal representation is used, but if negative, exponential notation is used. If a string variable, 'd' indicates the minimum width that the variable may take. If omitted, the default value for 'd' is 4 for switch variables, 2 for numerics, and 0 for strings; default for 'w' is the minimum required width. After the variable is inserted, it is subject to normal formatting operations. This will mean in many cases that all but the first leading space is ignored. Note that spaces may not occur between the &V, the variable name, and the parentheses.

&Wn: Wait (Pause)

The &W element causes execution of the display interpreter to pause for n tenth seconds. If 'n' is zero then a line of text (determined for each device, but normally "Press ENTER to Continue") is presented on the bottom line of the screen

or on the next line of the serial device. Execution then pauses until the student takes the action requested (usually pressing of the Carriage Return).

&Xnn Extended Character Set Access

For some devices (especially those that use an eight-bit character code), it is possible for the chosen input device to have a smaller character set than the output device on which the program is to be executed. These extra characters may be represented by the &Xnn control element where 'nn' indicates the specific character to be inserted. The string "&Xnn space" is replaced by the single indicated character. "nn" must be a two digit integer which has meaning within the FONT presently active. Similarly, &Xn where 'n' is a single digit may be defined. The single digit &Xn's should be reserved for positional characters such as Backspace, Index and Reverse Index.

&Y: Extensor

The &Y element is presently undefined. Its definition is reserved for future expansion, but if used by an installation, its meaning should be documented in any program in which it appears. It is recommended that &Y be used in conjunction with an indicative second character to remove restrictions on the number of expansion symbols. For example, &YA, &YB, ..., &YZ could be defined.

&Z: End AS-IS Mode

The first line of textual material following a set of AS-IS lines must begin with a control command; this command must begin with the &Z control element. &Z shifts out of AS-IS mode.

&-: Optional Hyphen

If &- appears within a word where it is desirable to break the word at the end of a transmitted line, it is replaced by a hyphen. Otherwise it is ignored.

&;: End of Display Statement

The &; element indicates the physical end of the Unit DISPLAY statement, in the same way that the semi-colon indicates the end of other statements.

4.3.2 Control Command Construction

Each control command begins with the current control character (represented by &). Following this is the body of the control command. The body consists of control elements, one after the other, with no intervening spaces. A space may follow the control command, and if so this space is part of the control command. The entire control command, including any trailing space is not a part of the textual material to be transmitted. If a non-space character terminates the control command, this character is taken as material for transmission. Control commands must contain at least one control element, but may contain more. Each control command is executed as it is encountered. If more than one element appears in the command, the elements are executed consecutively beginning with the left-most element. Some of the control elements require a single numeric modifier (for example &Kn). If this modifier is expressed as a literal, the element is formed by concatenation (for example &K3). If the modifier is input as a numeric variable, it must be placed in parentheses (for example &K(N)).

For those control elements which require parentheses (for example &C(r,c)) the modifiers may be either literal integers or variable names (examples &C(3,5), or &C(X,Y)). In all cases where variables are permitted, numeric expressions are also permitted. Where more than one numeric parameter is required, these may be supplied by a vector or pseudo vector.

For example, if the vector (5,5,20,15) were associated in the invocation of the Unit with the numeric vector name MID then the vector name could be used subsequently to name the area extending 20 rows down and 15 columns to the right of the character position (5,5). The control command element &E(MID) would erase this area; &B(MID) would cause subsequent text to appear in this area.

For those control elements that require a name of a variable (&V) or a function (&F and &G), this name must be a literal name. Argument lists for the &F and &G elements follow the standard rules for argument lists.

4.3.3 Text Conventions

One of the major text conventions was explained under &X control element. &X provides a means whereby characters not available on the input device may be designated for appearance on the output device.

A number of input devices do not have the capability of representing lower case characters. For such devices, a special convention for change of case is used:

the character '<' always indicates shift to upper case, and

the character '>' means shift to lower case.

The two symbols '<<' indicate a "literal" less-than symbol and '>>' indicate a literal greater-than symbol. For these devices, the upshift numeric and symbol characters may also be defined.

At times in the display of textual material, it may be desirable to have the current control command character appear. No special action is required if this character is to be followed by an illegal control command character (usually a space). However, a double representation (i.e.: "&&") will always cause a single control character to be inserted in the text stream.

Subscript and superscript characters may be specified in two ways. Special characters may be defined within a Font Block which appear smaller and in the upper or lower portion of the character display area; or the index and reverse index characters may be used to cause regular characters to be displayed on the next or previous line respectively. Both of these means depend upon the device used being able to accept and properly interpret the signals sent to it. No automatic means is provided for subscripting or superscripting on devices which are not capable of character redefinition or reverse-index function. For these devices the author may arrange to have subscripts or superscripts by doing his own text formatting in AS-IS mode.

The display processor does the major part of its text formatting by recognizing spaces as delimiters between words. In some cases, it is desirable to "fix" a single or multiple spaces between words. This is done by using the non-trivial space character: the underscore. For all formatting purposes, the processor treats the underscore as a regular non-space character. However, just before transmittal, all underscores are converted to space characters unless the underlining (&U) switch is on.

To the processor, a "word" is any character string which contains no space characters. Normally, the processor will fit as many of these words as possible into every line it transmits, and it will not break a word across two lines. The processor will break a word under two conditions: if the word is longer than a line, then the word is broken at the end of the line; or if the word contains a hyphen (minus sign). Words may contain either regular hyphens (eg: two-bit) or may contain "optional" hyphens denoted by "&-" (eg: con&-di&-tion). Normally, all optional hyphens are simply removed. However if a word will not fit at the end of a string, it is examined for hyphens. If it can be broken after a hyphen or an optional hyphen and still fit within the line, this is done. In the latter case, the optional hyphen is replaced by a hyphen.

The maximum display area or "page" is defined for each device type. The defined width of the page determines the maximum length of the lines to be transmitted. Many devices will have a maximum page length also. For these devices, the author may use &N control commands to do his own paging, or he can leave this task to the interpreter. By default, the interpreter will place as much display data as possible on the screen and then take an action

similar to an &N control element. It issues an &W0 and then advances or erases the screen as does the &N. It does not however cancel the &H,&M,&O, or &U.

Example

```
*MAMMALS : UNIT;
DISPLAY &N What is the main distinguishing feature of mammals? &;
RESPONSE TIME=20,CHARS=30;
EDIT MULSP,SHUP;
RIGHT CP('VIVIPAROUS')+CC('&N&EGG&')+CK('BRN LIVE');
FO    CC('&FUR&,&HAIR&');
WRONG (CC('&TEMP&')+CK('THRM'))*(CC('&SAME&')+CK('CNSTNT'))+CK('WRM BLD');
REINF RIGHT &L RIGHT! Mammals are viviparous and suckle their young. &;
REINF FO    &L It is true that only mammals are hairy. I was thinking
             more of the fact that mammals are viviparous. But I guess
             I'll let you by with the hair. &;
RETRY WRONG &L That holds for birds too. What does a mammal do that a
             bird doesn't do? &;
RETRY WRONG &L Oh, come now. Mammals bear live young and suckle them.
             &L Type VIVIPAROUS to continue. &;
RETRY UNREC &L I don't recognize that characteristic of mammals. Maybe
             this will help; what do female mammals do that other females
             don't do? &;
RETRY UNREC &L I don't think you have the right idea. What is different
             about the way mammals reproduce: &;
REINF UNREC &L Mammals bear live young and feed them milk from secretory
             glands on the mother's body. They are also warm-blooded and
             hairy. &L Let's go on. &;
END;
```

Sample executions of the above Unit

1. What is the main distinguishing feature of mammals?

*****intelligence

I don't recognize that characteristic of mammals. Maybe this will help: what do female mammals do that other females don't do?

*****preen

I don't think you have the right idea. What is different about the way mammals reproduce?

*****more enthusiasm

Mammals bear live young and feed them milk from secretory glands on the mother's body. They are also warm-blooded and hairy.

Let's go on.

2. What is the main distinguishing feature of mammals?

*****they're warm blooded

That holds for birds too. What does a mammal do that a bird doesn't do?

*****maintain a constant temperature

Oh, come on. Mammals bear live young and suckle them.
Type VIVIPAROUS to continue.

*****vivi#oops

I don't recognize that characteristic of mammals. Maybe this will help: what do female mammals do that other females don't do?

*****viviparous

RIGHT! Mammals are viviparous and suckle their young.

3. What is the main distinguishing feature of mammals?

*****homeothermy

That holds for birds too. What does a mammal do that a bird doesn't do?

*****grows fur

It is true that only mammals are hairy. I was thinking more of the fact that mammals are viviparous. But I guess I'll let you by with the hair.

4. What is the main distinguishing feature of mammals?

*****they don't lay eggs?

RIGHT! Mammals are viviparous and suckle their young.

Within the language, the tasks of general sequence control and collection, evaluation and storage of information are done within the Procedures. Procedures may also contain Condition Blocks which enable the definition of an environment within which the student receives his instruction.

Any Procedure may invoke any other Procedure, Unit, or Service Function. Entry Procedures are usually invoked by the system and are always the entry point of a Course. All other Procedures are invoked by the CALL or XFER statements or by the LINK System Procedure from other Procedures. Procedures may not be invoked from Units or Functions.

The statements permitted in Procedures are presented first in the section, followed by a discussion of the construction and use of the Condition Blocks. Construction and use of Procedures and Entry Procedures is presented next and discussion of a number of built-in system Procedures concludes the section.

5.1 PROCEDURE STATEMENTS

All of the statements presented in Section 3.1, (Function Statements), are permitted within Procedures. Those statements are: Assignment, Conditional, Null, DO, GOTO, and END. The EDIT statement (Section 4.1.5) is also permitted. Their use and meaning is identical within either Function or Procedures; they are not discussed in this section. The additional statements that are permitted in Procedures are discussed here. Unless otherwise indicated, all statements may be labelled statements.

5.1.1 The SIGNAL Statement

SIGNAL condition;

Special action to be taken when a certain "condition" arises is specified by CONDITION Blocks. The SIGNAL statement simulates the "Condition" and causes invocation and transfer of control to the Condition Block. The "Condition" specified in the SIGNAL statement must be a literal, and match that specified in the Condition Block. If the statement signals a Condition that has not been defined, or a Condition that has been disabled, then no action is taken.

5.1.2 The DISABLE Statement

DISABLE condition;

An established "condition" is rendered temporarily undefined by the DISABLE statement. If the Condition arises or is signalled but is currently disabled, then execution proceeds as if the particular Condition Block had not been defined. The specified "condition" must be a literal and match that specified in the Condition Block. If the specified "condition" is already disabled or is undefined, the DISABLE statement has no effect. The effect of a DISABLE statement is cancelled by a following ENABLE command specifying the same "condition" or when the routine in which the DISABLE statement appears reaches completion.

5.1.3 The ENABLE Statement

ENABLE condition;

A Condition Block, when executed, establishes special action to be taken whenever a specified condition arises. This action may be temporarily suspended by a DISABLE statement. In this case, the ENABLE statement is used to cause the Condition Block to revert to active status. The specified "condition" must be a literal, and match that specified in the Condition Block. If the "condition" specified is already active or if it has not been defined, then the ENABLE statement has no effect.

5.1.4 The CHECKPOINT Statement

CHECKPOINT;

A student taking a Course may either sign-off or be signed-off at any point in the execution. At some later time when he signs-on again, execution continues from a point at or just before where he was signed-off. These points are established automatically at the start of each Procedure and Unit, but further restart points may be specified within a Procedure by the CHECKPOINT Statement. The restart points established automatically are called Minor Restart Points. The CHECKPOINT statements identify Major Restart Points. These are discussed in section 5.7.

5.1.5 The CALL Statement

CALL name [(argument list)];

The CALL statement invokes another Unit or Procedure identified by "name", which is the label of the routine to be invoked. The label may be an indexed label, in which case "name" may take the form "name# (expression)". The "argument list" is constructed according to the rules for construction of argument lists, and must be enclosed in parentheses. When the Unit or Procedure that was called reaches completion, processing continues with the statement following the CALL statement. The parameters identified in the "argument list" may be altered by a called Procedure if they are unsubscripted variable names.

5.1.6 The XFER Statement

XFER name [(argument list)];

The XFER statement is similar to the CALL statement in that it invokes the Procedure labelled as "name". XFER may not be used to invoke a Unit or Function. XFER differs from CALL in that the named Procedure replaces the current Procedure; that is, the current Procedure becomes inactive and removed from the queue of Procedures and the named Procedure takes its place. Processing of the named Procedure commences with the first statement following the Header statement. The argument lists of the current Procedure and the Procedure named in the XFER statements must have the same number and type of arguments. If the XFER appears in an Entry Procedure, then the named Procedure must also be an Entry Procedure. Processing of this statement in an Entry Procedure causes the student to terminate the current Course and to begin the course named in the XFER statement. This registration information is altered to indicate that he is now taking a different course.

5.1.7 The REPEAT Statement

REPEAT [unit-name[(argument list)]];

The Repeat Statement provides an alternate method of invoking a Unit. Procedures may not be invoked by REPEAT statements. If "unit-name" is present and if the student has not previously executed the named Unit from the current Procedure, then the REPEAT behaves in the same way as

a CALL statement. If the student had previously executed the named Unit from the current Procedure, then execution proceeds as follows. The RESPONSE and REDITED system variables are cleared and the Unit is scanned for Reprise ENTRY statements. If one or more are found, the last such statement which has not been executed by previous invocations of the Unit is designated as the entry point. If all of the Reprise ENTRY statements had been previously executed, then the last one becomes the entry point. Execution of the Unit then proceeds as discussed under Section 4.2.1, Order of Execution. If the student has not previously executed the named Unit from the current Procedure, the Unit is executed from the beginning.

If "unit-name" is provided, then "(argument-list)" may be provided and the arguments passed in the normal way. The "(argument-list)" may be omitted if the Unit invocation does not require arguments, or if the Unit had been previously invoked. In the latter case, the argument list of the last invocation is used. If "unit-name" is omitted, the last Unit invoked from the Procedure is indicated. The "(argument-list)" may not be used. If no previous Unit was invoked by the Procedure, an ERROR condition is raised.

5.1.8 The STOP Statement

STOP;

If a STOP statement is encountered in execution of a Procedure, this signals that the student has reached the end of the Course. STOP raises the ENDCOURSE Condition. Execution is terminated and may not be restarted by normal restart means.

5.1.9 The EXIT Statement

EXIT;

If an EXIT statement is encountered in execution of a Procedure, a Major Restart point (CHECKPOINT) is automatically established and execution is terminated. Subsequently, if the student re-enters the course, execution will begin at the statement immediately following the EXIT statement.

5.1.10 The PROCTOR Statement

PROCTOR message;

The PROCTOR statement causes some text to be transmitted to a device identified by the installation as the Proctor Station. The text consists of the text identified by the system variable SPROC followed by the character representation of "message". "Message" may be null, a literal, a variable, or an expression; it is converted to a character string before being transmitted. If the Proctor Station has not been identified, the PROCTOR statement has no effect.

5.1.11 The RECORD Statement

RECORD item-list

The RECORD statement causes a record to be transmitted to a device identified by the installation as the Student Record device. The record consists of the items identified by the system variable SREC followed by the character representations of the items in the "item-list". "Item-list" consists of a series of literals or variables separated by commas. If a Student Record device is not identified or if the person executing the Course has not signed-on with a student code, the RECORD statement has no effect.

5.1.12 The SET Statement

SET S-variable <- S-list;

The SET statement is a pseudo-Assignment statement used to establish the value of certain global system action variables. "S-variable" may take any of the following forms:

SCOL: The SCOL pseudo variable is set by a SET statement to a character string which determines the collating sequence of characters. It must contain either the 54 characters of the Basic character set or the 95 characters of the Extended character set. Each character must appear exactly once in the string. Characters not in the Extended character set are treated as space characters. Accented characters, if not present in SCOL, are treated as being equal to their unaccented counterpart (for

example, "e" would be equal to "e"). Symbols, if not present in SCOL, are treated as equal to the space character. The initial setting is

```

b=+-*/( ),.:;&'#<>_!"$%?AaaaBbCcDdEeEeEeFfGfHh
IiJjKkLlMmNnOoOoPpQqRrSsTtUuUuVvWwXxYyZz0123456789

```

where *b* is the space character. This initial setting may be re-established by attempting to set SCOL to the Null string.

SCREEN: The SCREEN pseudo variable contains, in character form, two integer numerics indicating the maximum number of dots available on the current device in the horizontal (x) and vertical (y) direction respectively. The default setting of SCREEN is determined by the display device being used. SCREEN is NULL if the device is not capable of graphic display.

SEDIT: The SEDIT pseudo-variable may be SET to a list of Edit Function names separated by commas. These Edit-Functions act in the same way as an "edit-list" on an EDIT statement and determine the default or system editing of RESPONSE. The initial and default setting of SEDIT is "STANDARD".

SFONT: The SFONT pseudo-variable may be SET to the name of a Font Block. The name held by SFONT determines the default Font Block used for display of textual data. A system-supplied standard Font Block must be provided by the installation for each permitted device type. The name of this Font Block is the initial setting of SFONT.

SLANG: The SLANG pseudo-variable may be SET to a predominant user language indicator. Default setting is 'ENGLISH' or 'FRANCAIS' as determined by the installation.

SPROC: The SPROC pseudo-variable may be SET to a list of literals, pseudo-variables or system variables. The list of names in SPROC determine the prefix of all Proctor messages generated by a PROCTOR statement. Initially, SPROC is set to "IDevice,'" so that each Proctor message will be prefixed with the device number of the student causing the message. This part of SPROC is fixed and may not be altered, but further prefix information may be indicated by modification of this variable.

SREC: SREC is the same as SPROC except that it determines the prefix on performance recordings. Initial SREC setting is "WORD(QUEUE,1), SID,TIME2,WORD(QUEUE,-1),' '", to give the Course Name, Student Identification Code, current time and current Procedure or Unit name as a Performance Recording Prefix. The default portion of SREC may not be changed, but additional prefix information may be added to each performance recording by modification of this variable.

STIME: The STIME pseudo-variable may be SET to a literal numeric or numeric variable. Its value determines the default TIME for input represented by a RESPONSE statement. STIME has value 300.0 by default, indicating the default TIME is five minutes.

Any of the SET variables may be set in any Procedure. A setting takes effect at the time it is encountered, and remains in effect until reset by another SET statement or until the Procedure in which the variable was set reaches completion. When a Procedure reaches completion, the values of all SET variables set by that Procedure revert to the settings they had before the Procedure was invoked; i.e.: the set variable values are "stacked" by Procedure.

5.2 FILE ACCESS

There are numerous means already discussed by which output may be initiated. These include the Unit DISPLAY and Reprise statements, Graphic Functions (invoked by DISPLAY statements), and the PROCTOR and RECORD statements. The only means of input thus far identified has been the Unit RESPONSE statement. In this section, two further statements are discussed: READ and WRITE. These enable routines to accomplish direct-access file input and output.

5.2.1 Direct-Access Records

The author may open one or more direct access files for use in his Course by simply reading from them or writing to them, subject to restrictions imposed by the installation. Each file may contain a variable number of records; each record of varying length. Each record has an associated numeric key.

Records are placed in files at locations identified by a numeric key. The key ranges from 0 through +99,999.999 and has at most, three places of decimal. Each file also has two pointers called the Read Pointer and the Write Pointer. These pointers are changed by READ and WRITE operations, but at the first opening of a file, the Read Pointer is set at location one and the Write Pointer is set at the integer location immediately after the last record in the file. A READ statement specifying a Key of zero indicates the Read Pointer is to be positioned at the first record in the file.

5.2.2 The READ Statement

READ (filename [, [KEY=]key]) variable-list;

The statement must specify "filename", which is the name of the file from which records are to be read. If "key" is present then a direct READ is indicated; the Read Pointer is set to the location indicated by "key", the record is retrieved, and the Read Pointer is then set to point at the next higher location which contains a record. If "key" is not specified, a sequential READ is indicated; the record at the location where the Read Pointer is set is retrieved and the Read Pointer is set to point at the next higher location which contains a record. If there is no "next" record, for either type of READ, the Read Pointer is set to the end of the file. An attempt to do a Direct READ from a location which does not contain a record, or to do a Sequential READ when the Read Pointer is at the end of the file raises an ENDFILE condition for that file. The "key" parameter may be simply the key number or take the form "KEY=key number".

The "variable-list" is a list of unsubscripted variable names separated by commas. The record retrieved contains a variable number of values which are placed in the named variables on a one for one basis. The READ statement acts in a way similar to an Assignment statement with each variable in the "variable-list" on the left hand side and the corresponding retrieved value on the right hand side. The "variable-list" must not contain label variable names.

5.2.3 The WRITE Statement

WRITE (filename[, [KEY=]key]) expression-list;

The statement must specify "filename" which is the name of the file onto which records are to be written. If the file so named does not exist, the system will attempt to create it. If "key" is present, a Direct WRITE is indicated; a record composed of the values described by the "expression-list" is inserted at the location indicated by the "key". Writing a Null string at any location causes the record to be deleted. After writing, the Write Pointer is set to point at a location following the record.

This location is determined as follows:

- If no records after the last record written are present in the file, the Write Pointer is set to the next integer-valued location.
- If a record following the record just written is present, the Write Pointer is positioned midway between the record written and the next record in the file. If no locations are available between these two records, the Write Pointer is set to the next integer location after the last record in the file, and the INSERTION condition is raised at the next WRITE request.

If "key" is not present, a Sequential WRITE is indicated. The record is written at the location indicated by the position of the Write Pointer, and the Write Pointer is altered as discussed above. If this is the first record to be written in the file, and "key" has not been specified, the Write Pointer is set to location one before the record is written.

The "expression-list" consists of a list of expressions separated by commas. Each expression is placed in the record with a tag to indicate its type (logical, numeric, string, or vector), and its length. All expressions are evaluated and the literal results are used to construct the record. Label values may not be written to files.

5.3 COURSE RESOURCE CONTROL

In the normal execution of a course, each student will have a number of resources allocated specifically for his use. The actions of this student control and are controlled by these resources; the resources are not affected by other students executing the same or other courses. There are two particular resources, however, for which this is not the case. These are Course variables and files.

A particular course variable may be accessed by any number of students executing the same course. A particular file may be accessed by any number of students executing the same or different courses. A problem arises when two or more students require access to one of these "common" resources simultaneously.

Suppose an author has established the course variable "#NBODS" to count the number of students who take his course. Suppose further, that the following events take place:

1. Students A and B sign on to the same course.
2. Student A's course retrieves a copy of the current value of #NBODS and begins the necessary computation.
3. Meanwhile, student B's course retrieves the same current value of the variable and increments its value.
4. A's course puts the incremented value back.
5. B's course puts the (same) incremented value back.

Here, #NBODS should have been incremented by two, but wound up being incremented by only one.

The same problem could arise with files, and other problems can arise with both types of common resource. To prevent this, the course must "lock out" other students while it performs its computations. This locking out can occur in two ways: implicitly or explicitly.

Other students are implicitly locked out or prevented from using a common resource if the entire use of the resource is confined to a single assignment statement. That is, if the above incrementing had been specified by the author with the single statement

```
#NBODS <- #NBODS + 1;
```

then the system would have prevented other students' courses from accessing #NBODS until it had been properly incremented. This is the only means by which implicit lock-out of a common resource may be achieved. If more than one statement or a statement other than an assignment statement is required for updating course variables and for all updating of files, explicit lock-out must be used. This is accomplished using the HOLD and RELEASE statements.

5.3.1 The HOLD Statement

HOLD resource-list;

The HOLD statement prevents all other students from accessing those resources listed in the "resource-list". The "resource-list" is a list of names of files and course variables separated by commas. Course variable names are preceeded by the number symbol, which distinguishes them from file names. If any resource named is not available to the course containing the HOLD statement then that particular resource is not obtained, but all available named resources are obtained.

The resources obtained by a HOLD statement are "locked" so that no other user may access them. This lock remains in effect until:

1. Another HOLD statement is encountered; all items in the prior HOLD are released.
2. An implicit lock-out of a resource not named in the last HOLD is required; all items are released and a new HOLD is automatically issued to obtain previous items and the implicity required item.
3. A RELEASE statement is encountered; all items (if any) of the prior HOLD are released.
4. The routine reaches completion; same as RELEASE.

Any other course attempting to access any common resource which has been "locked" has its execution suspended until such time as all required resources are available. It is therefore important for the author to explicitly RELEASE common resources as soon and as often as possible.

5.3.2 The RELEASE Statement

RELEASE;

The RELEASE statement requires only the text "RELEASE;". It causes immediate release of all resources "locked" by a previous HOLD statement. Note that implicit RELEASE is indicated prior to any HOLD statement, any implicit "lock" and the completion of the current routine. If no resources are currently "locked" when a RELEASE statement is encountered, then the statement has no effect.

5.4 CONDITION BLOCKS

A Condition Block is a statement or a Do Block prefaced by an ON CONDITION statement. During execution of routines, a special circumstance may occur which causes an interruption of normal processing; this is called a "condition". The user may specify a group of statements to be executed whenever specified conditions occur.

5.4.1 The ON CONDITION Statement

ON CONDITION condition action

Like the Conditional statement, the ON CONDITION statement is not complete in itself. It must be followed by an "action"; a single statement or a Block of statements to be executed if and when the named "condition" arises. "Action" may be the Null statement indicating that no action is to be taken. Note that this is not necessarily the same as the action taken when the "condition" is disabled via a DISABLE command.

The presence of a Condition Block definition within a routine does not, in itself, ensure that that Condition Block will become active when the routine is executed. The Block becomes active only when it is encountered in the logical sequence of execution. The encounter of a Condition Block does not cause the Block to be executed. Execution can occur only when the Condition addressed by the Block is raised.

A Condition Block may appear at any location in any Procedure or General Function. When encountered, it becomes active and remains active until:

1. It is made inactive by a DISABLE statement.
2. It is overridden by the encounter of a new Condition Block addressing the same Condition.
3. The routine in which the Condition Block was made active reaches completion.

The ENABLE statement may re-activate a Condition Block only if 1 has occurred but 2 or 3 have not occurred. A Condition Block that has been overridden by encounter of a new Block in a lower-level routine addressing the same Condition becomes active when the routine in which the overriding Condition block is defined reaches completion.

5.4.2 Conditions

The various conditions that may arise during execution of routines are named by keywords. In the ON CONDITION statement, they identify the particular Condition to which the Condition Block applies. In some cases, they may be appended by localizing modifiers, which consist of a character string enclosed in parentheses. If a Condition keyword in an ON CONDITION statement does not have a modifier, it is taken to apply to all cases where the Condition is raised. If Condition Blocks are defined for both a modified and unmodified specific Condition which arises, the one containing the modified specification takes precedence.

ENDFILE or ENDFILE(filename): Whenever a READ statement is unable to retrieve the requested record, the ENDFILE Condition is raised. If "(filename)" is present then the Condition Block is executed only if "filename" matches the name of the file from which the record could not be obtained. Default system action is to abnormally terminate processing (ABEND).

INSERTION or INSERTION(filename): Whenever a WRITE statement cannot write a record because of no appropriate location in the file, the INSERTION Condition is raised. If "(filename)" is present then the Condition Block is executed only if "filename" matches the name of the file being written. Default system action is ABEND.

RESTART: Whenever a student signs-off or is signed-off and subsequently recommences execution of a Course, the system goes through a procedure called a Restart. This consists of restoring the students execution environment to a state essentially the same as that from which he last signed-off. After the Restart procedure, but just before execution recommences, the RESTART Condition is raised. Default system action is to take no special action.

ENTRY or ENTRY(name): Whenever control passes via a CALL statement from a Procedure to a Procedure or to a Unit, the ENTRY Condition is raised. If "(name)" is present, then the Condition Block is executed only if "name" matches the routine to which control is being transferred. Default system action is to take no special action. Note that the RENTRY Category occurs only after the ENTRY Condition is raised by a RETRY statement.

ERROR or ERROR(n): Whenever an execution error occurs, the ERROR Condition is raised. Examples of execution errors are: DIVISION by zero, OVERFLOW and UNDERFLOW where results of a calculation are too large or too small to be expressed internally, INPUT or OUTPUT errors other than ENDFILE or INSERTION such as an illegal key or too large a record size, and VECTOR element access errors. Each specific error has an associated integer value which is placed in IERR when the error occurs. The modifier "(n)" where "n" is a literal integer may be used to trap the specific error of that number. Default system action varies but is usually ABEND.

RESPONSE or RESPONSE(unit-name): Immediately after receipt of a student response as requested by a RESPONSE statement, the RESPONSE Condition is raised. The "(unit-name)" modifier may be used to restrict the Condition Blocks range to a specific Unit. Default system action is to take no special action.

RTYPE or RTYPE(unit-name): When a student response is requested by a RESPONSE statement, the author may specify the TYPE of response anticipated. If the student's response is not of that TYPE, the RTYPE Condition is raised. The "(unit-name)" modifier may be used to restrict the range of the Condition Block to a single Unit. Default system action is to scan following Unit statements for a Reprise RTYPE statement. If one is found, it is executed. Otherwise the default action is to print a device-dependent message to the student and return to the RESPONSE statement.

OTIME or OTIME(unit-name): The author may specify a time limit within which the student is to complete a response. If this time limit is exceeded, the OTIME Condition is raised. The "(unit-name)" modifier may be used to restrict the range of the Condition Block to a single Unit. Default system action is to accept the (perhaps incomplete) student response and continue normal processing.

ATTN: On some devices, a key is provided to indicate a user request to interrupt processing. Normally, this key is disabled, but the author may enable the key and specify the action to take when this key is depressed by providing a Condition Block with the ATTN Condition.

Cat-Condition or Cat-Condition(unit-name): Categorization statements may set the system switches FRIGHT, FWRONG, F0, F1, F2, ... F9. Each time one of these switches is set, a corresponding Condition named RIGHT, WRONG, F0, F1, ..., F9 is raised. If a system switch is not set by any of the categorization statements in a Unit, the FUNREC switch is set and the UNREC condition raised. The "(unit-name)" modifier may be used to restrict the range of the Condition Block to a single Unit. Default system action is to take no special action.

UNREC or UNREC(unit-name): During execution of a Unit, if a Reprise statement addressing the UNREC Category is encountered or if the END statement is encountered, the category switches F0, F1, F2, ... F9, FRIGHT, and FWRONG are examined. If none of them are set, the UNREC Condition is raised. The "(unit-name)" modifier may be present to restrict the scope of the Condition Block to a single unit. If the UNREC Condition was raised by a Reprise statement, the UNREC Category occurs and default system action is

to continue with the execution of the Reprise statement. Otherwise, default system action is to take no special action.

INPUT('string'): Under the same circumstances, but just before the UNREC condition is raised, the student response is compared to all "string"s specified on INPUT Conditions. If it matches one of these, the INPUT Condition corresponding to that "string" is raised. The "('string')" must be specified; the INPUT Condition has meaning only in terms of its specified "string". Default system action does not apply; no special action is taken if a student response does not match any INPUT Condition "string".

ENDCOURSE: When sequence of execution reaches the logical end of the Entry Procedure for a Course, when a STOP statement is encountered, when transfer (XFER) is made to an Entry Procedure, or when a Procedure which is not an Entry Procedure but which is the highest controlling Procedure (established by a XFER) reaches completion, the ENDCOURSE Condition is raised. Default system action closes the student's records for that course and then either sign-off or commence the new Course.

5.4.3 The Extended RETURN Statement

RETURN[location];

Within Condition Blocks defined in Procedures, the extended form of the RETURN Statement may be used to control the sequence of execution after the Condition has been raised and the Condition Block executed.

The "location" may be any of the following:

RESTART: Indicates that execution is to begin again at the last Restart point passed.

CALLER: Indicates that the current routine execution is to be suspended and return made immediately to the last invoked pending.

ENTRY: Indicates that execution is to proceed with a Reprise RENTRY statement in the current or last executed Unit. The point for commencement of execution is determined in the same way as if a REPEAT statement had been encountered.

REPEAT: Indicates that execution is to continue with the statement that raised the Condition.

CONTINUE: Indicates that execution is to continue with the statement following the one which raised the Condition.

If "location" is omitted, then a default "location" is assumed. For the ENDFILE, INSERTION, RESTART, and ERROR Conditions, RESTART is the default "location"; for the INPUT Condition, ENTRY is the default "location"; for ENDCOURSE, STOP is assumed; for all other Conditions, CONTINUE is the default "location". If sequence of control passes through the logical end of the Condition Block without encountered a RETURN statement, then a default RETURN is taken.

5.4.4 Condition Block Construction and Execution

All Condition Blocks consist of an ON CONDITION statement which identifies the Condition to which the Block applies and serves as a Header for the "action" that is to take place in the event that the Condition is raised. The "action" may consist of either a single statement or a DO-Block.

If the "action" consists of a single statement, this statement may be any of the following statements: Assignment, Null, READ, WRITE, PROCTOR, RETURN, REPEAT, EDIT, RECORD, CALL, XFER, EXIT, or STOP. The Null statement indicates that no action is to be taken. Immediately after executing any single statement which does not indicate a change of control, return is made to the default return point for the particular Condition.

The single statement may also be a GOTO statement specifying a location within the routine in which the Condition Block is defined. Execution of this statement causes all pending routines below this routine to become inactive, the Condition Block to terminate, the routine containing the Condition Block definition to become active, and execution to commence at the location specified on the GOTO statement.

If the "action" does not consist of a single statement, it must consist of a single DO-Block. This DO-Block must be headed by one of the three forms of the DO-Block Header. Within this DO-Block, the following additional statements are permitted: Conditional, DO (all forms), GOTO, the Extended RETURN, and END. If the GOTO specifies a location within

the Condition Block, normal processing continues; if a location outside the Condition Block, this location must be within the Procedure in which the Condition Block is defined. Execution proceeds as discussed with GOTO as a single statement Condition Block.

Sequence of execution may or may not pass out of the Condition Block, but the Block has current access to all variables within the routine in which it is defined and to system variables. The Block does not have access to the variables of the routine in which the Condition is raised unless it is the same routine in which the Condition was defined.

When a Condition is raised, a search is made for an active Condition Block pertaining to that Condition. If one is found, the Condition is cancelled and the Block is immediately executed. If one is not found, the Condition is immediately cancelled. The invocation of a Condition Block is essentially the same as a CALL to a Procedure. The Block may CALL other Procedures or Units, and remains in a pending state until all processing resulting from its invocation has been completed. Recursion (the raising of a Condition while the Condition Block addressing that Condition is pending) is permitted; pending Condition Blocks are stacked in the same way as pending Procedures.

The INPUT ('string') form of condition specification may be used to establish traps for certain meaningful words which the student may enter to alter sequence of execution. While the types of actions an author may wish to permit and the associated words may exhibit considerable variation, the following standard words should be used with their given actions:

CALC: Causes the CALC Procedure to be invoked.

HELP: Takes a general (not specific to the particular student's current problem) remedial action such as informing the teacher.

GLOSSARY: Invokes a routine allowing the student to retrieve definitions of words.

COMMENT: Permits a student to enter a comment of any kind to be sent to the author or instructor.

APPEAL: Permits the student to indicate that he feels his response was correct when the program classified it otherwise.

Condition Block Examples

1. ON CONDITION UNREC
RECORD RESPONSE;
2. ON CONDITION RESTART
DO;
WRITE (RFILE) SID,DATE,TIME2;
PROCTOR 'RESTART',SID;
END;
3. ON CONDITION INPUT('GLOSSARY')
DO;
N1 <- MINUTES(TIME2);
CALL GLOSSARY;
N1 <- MINUTES(TIME2) - N1;
IF (N1 <= 0.5) THEN RETURN REPEAT;
IF (N1 <= 5) THEN RETURN ENTRY;
ELSE RETURN RESTART;
END;

5.5 THE GENERAL PROCEDURE

5.5.1. Purpose of the General Procedure

General Procedures are the controlling routines and the building blocks of Courses. They assemble the data required by subordinate routines, determine which and in what order Units and other Procedures are to be invoked, collect and save information generated as the result of the processing of subordinate routines, and establish the environment of the Course.

5.5.2 Invoking a General Procedure

A General Procedure is invoked by the appearance of its name (the label on its Header statement) in a CALL or XFER statement. Procedures are also invoked by the system Procedure, LINK. Between the time a Procedure has been invoked and the time an XFER, RETURN, EXIT or STOP statement or the logical end of the Procedure is encountered, the Procedure is said to be "pending".

When the Procedure is invoked, those variables present in the argument list on the Header statement are initialized by the calling Procedure according to the rules for argument lists. Any changes made to these variables are maintained in the calling Procedure when control is returned.

5.5.3 Constructing a General Procedure

The General Procedure may employ the following statements: Procedure Header, Assignment, Conditional, Null, RECORD, PROCTOR, READ, WRITE, DO GOTO, RETURN, CALL, XFER, REPEAT, ON CONDITION, SIGNAL, ENABLE, DISABLE, CHECKPOINT, SET, STOP, EXIT, and END. The General Procedure must begin with a General Procedure Header statement and end with a corresponding END statement.

The Header statement of the General Procedure is constructed as follows:

```
label: PROCEDURE[(argument-list)];
```

The General Procedure Header statement must be a labelled statement. The label may be a standard label or a literal indexed label. The Procedure is known to the system and invoked by other Procedures via the label on its Header statement. The "argument-list" follows the rules for construction of argument lists (Section 2.3.5) and must be enclosed in parentheses. The "(argument-list)" may be omitted if no parameters are required. The Procedure may consider the variables represented in the argument list to be initialized when the Procedure is invoked. Any changes made to these variables are maintained and transferred to the corresponding variables in the calling Procedure when control is returned.

Example

```

GLOSSARY : PROCEDURE;
/*
* THIS PROCEDURE CONTROLS EXECUTION OF THE GLOSSARY
* ROUTINE. STUDENT IS ASKED TO ENTER A WORD AND A
* SEARCH IS MADE OF THE DEFINITIONS (DFILE). IF
* FOUND, THE DEFINITION IS GIVEN. THE STUDENT MAY
* ENTER "BROWSE XXX" TO ALLOW SEQUENTIAL (ALPHABET-
* ICAL) ACCESS TO DEFINITIONS. BROWSING STARTS AT
* THE FIRST WORD BEGINNING WITH XXX.
*/
BEGIN:
  READ (DFILE,KEY=0);          /* REWIND DFILE */
  CALL *GLOSSARY_REQUEST;
  IF (F0) THEN GOTO BROWSE:    /*REQUEST TO BROWSE */
  IF (F1) THEN RETURN;        /* REQUEST TO QUIT */
  REQ <- WORD(RESPONSE,1);
SEARCH:
  READ(DFILE) TERM,REPLY;
  IF ( TERM < REQ ) THEN GOTO SEARCH;
  IF ( TERM = REQ ) THEN CALL *GLOSSARY_REPLY(TERM,REPLY);
  ELSE CALL *GLOSSARY_NOTFOUND;
  GOTO BEGIN;
BROWSE:
  REQ <- WORD(RESPONSE,2);
  IF (LEN(REQ) = 0) THEN DO;    /* XXX MISSING */
  REPEAT; /*REMIND ABOUT XXX*/
  GOTO BEGIN;
  END;
SCAN:
  READ (DFILE) TERM,REPLY;
  IF (TERM < REQ) THEN GOTO SCAN;
SHOW:
  CALL *GLOSSARY_REPLY(TERM,REPLY);
  IF (F1) THEN GOTO BEGIN; /*REQUEST TO STOP BROWSE*/
  READ (DFILE) TERM,REPLY;
  GOTO SHOW;
END;

```

Units required by the GLOSSARY Procedure

```

*GLOSSARY REQUEST: UNIT;
  DISPLAY &N Enter &U GLOSSARY &U Request: &;
  RESPONSE;
  EDIT SHUP;
  FO CC('BROWSE&')
  F1 CC('QUIT')
  REINF F1 &L END OF GLOSSARY &L Press ENTER to continue. &N;
  REINF ENTRY &L Please provide a "start-string" &L
    Enter "BROWSE xxx" where "xxx" is the
    first word whose definition you wish
    to see. &;
END;

```

```

*GLOSSARY REPLY: UNIT(TERM,REPLY);
  DISPLAY &L2U &V(TERM) &UL &V(REPLY) &;
  RESPONSE CHARS=1;
  FO CC(' ')+CC('');
  F1 -FO;
END;

```

```

*GLOSSARY NOTFOUND: UNIT;
  DISPLAY &L SORRY, I don't have a definition for that
    word. Please try a synonym. &;
END;

```

5.6 THE ENTRY PROCEDURE

5.6.1 Purpose of the Entry Procedure

The purposes of the Entry Procedure include those of the General Procedure. In addition, the Entry Procedure serves as the entry point and designates the name of a Course. The ON CONDITION and SET statements establish default actions to occur in all execution that follows. Whenever possible, these statements should be included in the Entry Procedure; to the extent this is done, the Entry Procedure has the further purpose of establishing the environment in which the remainder of the Course routines are executed.

5.5.2 Invoking an Entry Procedure

Entry Procedures are normally invoked by the system in response to a request by a student to execute a Course of the same name as the label on the Entry Procedure Header statement. Note that an Entry Procedure may not have an indexed label. Entry Procedures may also be invoked from other Procedures by the CALL or XFER statements or the LINK system Procedure. If the Entry Procedure is invoked by an XFER statement from another Entry Procedure, all prior pending Procedures and Condition Blocks are released. In addition, the student records are modified to indicate the new Course he is taking. If the Entry Procedure is invoked by a CALL statement, the LINK System Procedure, or by an XFER from a non-Entry Procedure, then the Entry Procedure behaves in the same way as any other Procedure. Whenever execution control reaches the logical end of the Course Entry Procedure, the student is considered to have finished the Course and the ENDCOURSE Condition is raised.

5.6.3 Constructing an Entry Procedure

The Entry Procedure may employ any of the statements permitted in General Procedures. It must begin with an Entry Procedure Header statement and end with a corresponding END statement. The Header statement for an Entry Procedure is constructed as follows.

```
label: ENTRY PROCEDURE;
```

The Entry Procedure Header statement must be a labelled statement; the label may not be an indexed label. Parameters may not be passed to Entry Procedures; an argument list is not permitted on the Header statement. The label becomes the name of a Course which consists of the Entry Procedure and all necessary subordinate routines.

Example

```
BROWSE : ENTRY PROCEDURE;
/*
* THE BROWSE COURSE ALLOWS A USER TO "BROWSE" THROUGH THE
* UNITS AND PROCEDURES AVAILABLE IN THE SYSTEM. IF HE
* ENTERS THE TEXT "BROWSE" AT ANY TIME, HE RETURNS TO THE
* BROWSE PROCEDURE. THE NORMAL EXPLAIN, GLOSSARY, AND
* INDEX PROCEDURES ARE ENABLED.
*/
ON CONDITION ERROR          CALL *BROWSE_ERROR;
ON CONDITION INPUT('BROWSE') XFER BROWSE;
ON CONDITION INPUT('EXPLAIN') CALL EXPLAIN;
ON CONDITION INPUT('GLOSSARY') CALL GLOSSARY;
ON CONDITION INPUT('INDEX')  CALL *INDEX;
    CALL EXPLAIN_BROWSE;
GET_A_LABEL:
    CALL *BROWSE_GET;
    ROUTE <- WORD(RESPONSE,1);
    IF (ROUTE = 'QUIT') THEN STOP;
    N <- LEN(ROUTE);
    SPARS <- NULL;
    IF (N < LEN(RESPONSE)) THEN SPARS <- SUBSTR(RESPONSE,N+1);
    CALL LINK(ROUTE,SPARS);
    GO TO GET_A_LABEL;
END;
```

Units Required by the BROWSE Entry Procedure

```
*BROWSE ERROR: UNIT;  
  DISPLAY &L2 *****ERROR***** &L2 An error occurred while attempting  
    to LINK to a Unit or Procedure or while executing a subordinate  
    routine. Type "BROWSE" to exit, or just ENTER to continue. &;  
  RESPONSE APPEND;  
  END;
```

```
*BROWSE GET: UNIT;  
  DISPLAY &L ENTER the name of the Unit or Procedure you wish to execute  
    followed by the required parameters separated by spaces. &;  
  RESPONSE;  
  EDIT SHUP,MULSP;  
  END;
```

5.7 SYSTEM PROCEDURES

5.7.1 Dynamic Invocation Procedure (LINK)

LINK(name[,argument-list])

The LINK System Procedure is provided to enable the user to invoke another Unit or Procedure whose name is determined at execution time. "Name" must be a literal string or a string variable containing the name of an existing Procedure or Unit. The "argument-list" must be constructed according to the rules for construction of argument lists, and is the argument list passed to the Unit or Procedure invoked by LINK. LINK is invoked by either a CALL or XFER statement. An ERROR condition is raised if the label provided in "name" does not match that of any existing Units or Procedures, or if LINK attempts to invoke itself. If the routine being invoked has an indexed label, then "name" must contain a literal version of that label. The LINK Procedure does not enter the pending state and is not included in the QUEUE system variable. Its actions are transparent to sequence of execution which proceeds as if the routine had been invoked by a direct CALL or XFER statement. The LINK Procedure is used in the example in Section 5.6.3.

5.7.2 Student Calculation Mode (CALC)

CALC [(variable-list,parameter-list)]

The CALC system Procedure places the student in a mode where he can perform simple arithmetic computations. The "variable-list" parameter must be present if an argument list is provided. It may be a literal

string on a string variable containing a list of single-letter names separated by commas. Numeric variables of the same names are made available to the student for his computations. If "variable-list" is Null, the letter names 'X', 'Y', and 'Z' are made available for use as variable names. The remainder of the argument list consists of normal argument list elements which correspond to the variables made available to the student. For example, "CALL CALC ('A,B,C',2,2**5,N)" invokes the Calculation Mode, making variables A,B, and C available to the student. "A" would initially have the value 2, "B" the value 32, and "C" the value of "N". When control returns, "N" maintains the value left by the student in his variable "C".

"Parameter-list" contains only numeric elements; each element of a vector constitutes an element of "parameter-list". The association between the variable names in "variable-list" and the elements of "parameter-list" are made on a one for one basis from left to right. Any variable names not associated with an element are initialized to zero. Only variable names associated with Procedure variables may have their values retrieved by the Procedure.

In Calculation Mode, the student has access to only the non-vectored numeric variables names in "variable-list". He may enter expressions, assignment statements, or the completion message "DONE". Expressions consist of one or more literal numerics and/or variable names separated by operators. The operators permitted and their symbols are sum(+), difference(-), product(*), quotient(/) and power(**). The sum and

difference operators may be used as unary prefix operators; all operators express their normal arithmetic operation request. Parentheses may be used to group operations in the conventional way. The functions SIN(X), COS(X), TAN(X), SQRT(X), LOG(X), LOGE(X) AND ABS(X) may be employed and result in the sine, cosine, tangent, square root, base ten logarithm, natural logarithm and absolute value of the single numeric literal or variable argument "x" respectively. Normal restrictions on the range of x apply and violations produce an appropriate error message. When an expression is entered by the student, it is evaluated and the result printed before further input is accepted.

Assignment statements are identified by the presence of the assignment symbol "<=". They must have a single acceptable variable name at left of the assignment symbol and an acceptable expression at right of the symbol. When a student types an assignment statement, the expression on the right is evaluated and the result placed in the variable named on the left. Nothing is printed.

When invoked, CALC prints "***CALC MODE:" followed by a list of the permitted variables. CALC Mode remains in effect until the student responds with the completion message "DONE". The CALC Procedure returns control in the same way as any other Procedure.

5.7.3 Compiler and Interpreter Invocation

External compilers and interpreters (for example FORTRAN, COBOL, PL/I, APL, BASIC, FOCAL) may be accessed from the language by means of special system Procedures. These Procedures have a name formed from the name of the compiler or interpreter being invoked. This name may include a designation to distinguish among two or more versions of the same compiler or interpreter. For example, the names FORTG and FORTH would be used to invoke the FORTRAN G and H compilers respectively. Invocation is accomplished through any of the normal Procedure invocation means. For example:

```
CALL name(argument-list);
```

The "argument-list" contents and format and the particular interpreters and compilers available are determined by each installation according to the facilities they have available.

INTERPRETERS

For those interpreters embedded in systems such APL and BASIC, a call to the system Procedure with no parameters indicates that the system is to be invoked. For example,

```
CALL APL;
```

invokes the APL system allowing the student all normal APL facilities such as function definitions and workspace SAVE. Upon termination of APL however, control returns to the statement following the CALL statement. Variables may be passed to and retrieved from interpretive systems by means of the argument list in the same way as with the CALC system Procedure.

COMPILERS

For those compilers (usually in-core compilers) providing dynamic execution facilities (for example WATFIV), invocation without the specification of a parameter list indicates that the student will provide input directly to the compiler, request execution in the normal way, and receive output on the default device. If an argument list is provided it must take the following form:

(par1, par2, par3, par4)

Where "par1" is the name of a file supplying source input to the compiler, "par2" is the name of a file receiving compiler diagnostics and output, "par3" is the name of a file of data for input at execution time, and "par4" is the name of a file receiving any printout resulting from program execution. If "par1" is omitted, compiler output is printed on the default device as received; if "par3" is omitted, data is requested (if required) from the student; if "par4" is omitted, program execution output is printed on the default device as received.

For those compilers which do not provide for dynamic execution, an argument list must be provided on the invocation statement. It takes the form:

(par1, par2, par5)

Where "par1" and "par2" are source input and compiler output files as above and "par5" is the name of a file into which the object or executable form of the compiled program is placed. If "par5" is omitted, the compiled

program is not saved. The content of the file named by "par5" may be loaded and executed by the system Procedure, EXEC.

EXEC(par5, par3, par4)

Where "par5" is the name of the file where an object or executable code deck has been placed, "par3" is the name of a file of data for input during execution and "par4" is the name of a file to which output from execution is to be directed. "Par5" may not be omitted. If "par3" is omitted, data is requested (if required) from the student; if "par4" is omitted, output is printed on the default device as received.

5.8 Restart Conventions

During execution of a Course, a student may either intentionally signal that he wishes to discontinue, or be forced to discontinue by a supervisor (intentional sign-off); he may also be forced to discontinue through some type of computer error or mishap (unintentional sign-off). At a later time, when he signals that he wishes to continue work again, a procedure called RESTART is undertaken to re-establish the students environment.

A Major Restart is required when the student intentionally signs-off, when he is signed-off by the supervisor or an EXIT statement, or when he is unintentionally signed-off and does not sign-on again within ten minutes of sign-on becoming possible. A Minor Restart is indicated when the student is inadvertently signed-off but signs back on again as soon as he is able.

When a Minor Restart occurs, the student's environment is established and execution commences as if a REPEAT statement were encountered for the last Unit began or completed by the student. Major Restart always commence at the point indicated by the last CHECKPOINT statement encountered by the student. Note that if a Condition Block has been established for the RESTART Condition, it is invoked for both types of Restart, and its requests are completed before normal execution continues. If the location to which the Condition Block transfers control on completion is defaulted, it is established by the Major or Minor Restart point.

5.9 Synonym Facility

In the construction of any routine, the Header statement may be immediately followed by a Synonym Block which consists of one or more SYN statements.

5.9.1 The SYN Statement

SYN name 'string'

The SYN statement associates a "string" with a user-determined "name". The string must be constructed according to the rules for construction of literal strings. It may be a null string. The space is a valid and legal character in "string". A single apostrophe character may be represented in "string" by two consecutive apostrophe characters.

The "name" should be chosen so as not to conflict with the names of variables used in the routine in which the SYN statement appears. It must be constructed of the alphabetic, digit and underscore characters, and must begin with an alphabetic. Wherever "name" appears it is replaced by "string" in the expansion of the routine. This expansion occurs prior to execution. SYN "name"s should not appear within SYN "string"s; if this happens, the name is not recognized as a SYN "name".

The SYN statement may be used to cause statement components, statements, or groups of statements to be associated with a single name. In the statements following the Synonym Block, the single name may be used whenever the entire contents of the "string" are required.

5.9.2 SYN Examples

1. BOX : GRAPHIC FUNCTION:

```
SYN LINE '*FFFFFFFFFF';
SYN SIDE '*8000000001';
```

```
MAP (10,40)  HEX
LINE
SIDE SIDE SIDE SIDE
SIDE SIDE SIDE SIDE
LINE;
```

END;

The above Graphic Function, after expansion, would be equivalent to the following:

[illegible]

The task accomplished by the above function is to display a box which is about one character in height and about six characters in length. It would be done more easily using the PLOT statement. The MAP statement was used here to illustrate the effect of the SYN statement.

2. The following Unit call Graphic Functions VOLKSWAGEN, HAMMER, and TOP which display pictures of a Volkswagen, a hammer, and a top respectively. Each of these require four parameters identifying the screen area in which the picture is to be displayed.

*ELEMENTARY_COLLECTIVE_NOUNS : UNIT;

```

SYN TEXT '1,1,40,20';
SYN CAR  '1,21,10,20';
SYN TOOL '11,21,10,20';
SYN TOY  '21,21,10,20';

```

DISPLAY &L TEXT At the right are three pictures. First is a picture of a &U Volkswagen &U, then a &U hammer &U, and then a &U top &U. Take the light-pen, and point to the picture of a TOY.

&G(VOLKSWAGEN(CAR), HAMMER(TOOL), TOP(TOY)) &;

RESPONSE TYPE=C;

RIGHT CG(TOY);

REINF RIGHT &L RIGHT. A top is one kind of toy. &;

F1 CG(CAR);

RETRY F1 &L No, a car is not a toy. Try again. &;

F2 CG(TOOL);

RETRY F2 &L A hammer isn't really supposed to be used as a toy. Point to the picture of something which can only be used as a toy. &;

RETRY UNREC &L Please point to one of the three pictures. &;

END;

Two uses of the SYN statement are illustrated in the above example. One is that a meaningful name may be associated with a location on the screen; the second is that the location is specified only once and then referred to by name twice in the construction of the Unit. The meaningful names help both the original creator and subsequent readers of the Unit to understand what is going on. They may also reduce the amount of coding required by the author.

APPENDIX A

LANGUAGE KEYWORDS AND SYMBOLS

(In those cases where the French Language Keyword differs from the English, it is shown in parentheses)

Keywords

APPEND (AJOUTER)	Keyword used on the RESPONSE statement.
AT (A)	Keyword used on the MAP statement.
ATTN (ATT)	Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements.
BACKSPACE (RECU)	Terminal operation name used on CHARACTER statement.
BIN	Keyword used in specification on MAP statement.
BY (PAR)	Keyword used in specification on DO statement.
CALL (APPELER)	Keyword of the CALL statement.
CALLER (DEMANDEUR)	Specifiable destination on extended RETURN statement.
CHARACTER (CARACTERE)	Keyword of the CHARACTER statement.
CHARS (CAR)	Keyword used on the RESPONSE statement.
CHECKPOINT (CONTROLE)	Keyword of the CHECKPOINT statement.
CONDITION	Recognizable portion of the ON CONDITION keyword.
CONTINUE (CONTINUATION)	Specifiable destination on extended RETURN statement.
DISABLE (INVALIDER)	Keyword of the DISABLE statement.
DISPLAY (AFFICHER)	Keyword of the DISPLAY statement.
DO (FAIRE)	Keyword of the DO statement.
EDIT (EDITER)	Keyword of the EDIT statement. Function type name used on the Header statement.
ELSE (SINON)	Keyword used on the Conditional statement.
ENABLE (VALIDER)	Keyword of the ENABLE statement.
END (FIN)	Keyword of the END statement.
ENDCOURSE (FINCOURS)	Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements.

ENDFILE (FINFICHIER)	Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements.
ENTRY (ENTREE)	A Procedure type (ENTRY PROCEDURE) used on Header statement. A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Specifiable destination on the extended RETURN statement.
ERROR (ERREUR)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements.
EXIT (SORTIE)	Keyword of the EXIT statement.
F0, F1, F2, F3, F4, F5, F6, F7, F8, F9	System switch names. Category names used on the Category statement. Condition names used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Category names used on the Reprise statement.
FALSE (FAUX)	Literal switch value zero.
FONT (JEU)	Keyword used on the RESPONSE statement.
FONT BLOCK (BLOCJEU)	A routine type used on the Header statement.
FUNCTION (FONCTION)	A routine type used on the Header statement.
GO (ALLER)	Identifiable portion of the GOTO keyword.
GOTO (ALLERA)	Keyword of the GOTO statement.
GRAPHIC (GRAPHIQUE)	A Function type (GRAPHIC FUNCTION) used on the Header statement.
HEX	Keyword used in specification on the MAP statement.
HOLD (RESERVER)	Keyword of the HOLD statement.
HOME (DEPART)	Terminal operation name used in the CHARACTER statement.
IF (SI)	Keyword of the Conditional statement.
INDEX (DEMILIGNE INF)	Terminal operation name used on the CHARACTER statement.
INPUT (INTRODUCTION)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements.
INSERTION	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements.
KEY (CLE)	Keyword used on the READ and WRITE statements.
LINEFEED (INTERLIGNE)	Terminal operation name used on the CHARACTER statement.

MAP (DESSINER SOUS)	Keyword of the MAP statement.
ON (SUR)	Identifiable portion of the ON CONDITION keyword. Keyword used in the DISPLAY statement.
ON CONDITION (SOUS CONDITION)	Keyword of the ON CONDITION statement.
OTIME (SURTEMPS)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Category name used on the Reprise statement.
PLOT (TRACER)	Keyword of the PLOT statement.
POSN (POS)	Keyword used on the RESPONSE statement.
PROCEDURE (PROCEDURE)	A routine type used on the Header statement.
PROCTOR (CENTRAL)	Keyword of the PROCTOR statement.
READ (LIRE)	Keyword of the READ statement.
RECORD (ENREGISTRER)	Keyword of the RECORD statement.
REINF	Action category name used on the Reprise statement.
RELEASE (LIBERER)	Keyword of the RELEASE statement.
RENTY (RENTREE)	Category name used on the Reprise statement.
REPEAT (REPETER)	Keyword of the REPEAT statement. Specifiable destination on the extended RETURN statement.
RESPONSE (REPOSE)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Keyword of the RESPONSE statement.
RESTART (RELANCE)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Specifiable destination on the extended RETURN statement.
RETRY (RESSAYER)	Action category name used on the Reprise statement.
RETURN (RETOUR)	Terminal operation name used on the CHARACTER statement. Keyword of the RETURN statement.
REVERSE INDEX (DEMILIGNE SUP)	Terminal operation name used on the CHARACTER statement.
RIGHT (EXACT)	Category name used on the Category statement. A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Category name used on the Reprise statement.

RTYPE (TYPEREP)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Category name used on the Reprise statement.
SET (FORCER)	Keyword of the SET statement.
SIGNAL (SIGNALER)	Keyword of the SIGNAL statement.
STOP (ARRETER)	Keyword of the STOP statement.
SUBSCRIPT (INDICEINF)	Terminal operation name used on the CHARACTER statement.
SUPERScript (INDICESUP)	Terminal operation name used on the CHARACTER statement.
SYN	Keyword of the SYN statement.
THEN (ALORS)	Keyword used on the Conditional statement.
TIME (TEMPS)	Keyword used on the RESPONSE statement.
TO (JUSQUA) (A)	Keyword used in specification on a DO statement. Identifiable portion of the GOTO keyword.
TRUE (VRAI)	Literal switch value one.
TYPE	Keyword used on the RESPONSE statement.
UNIT (UNITE)	A routine type used on the Header statement.
UNREC (NONREC)	A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Category name used on the Reprise statement.
WHILE (TANTQUE)	Keyword used in specification on a DO statement.
WRITE (Ecrire)	Keyword of the WRITE statement.
WRONG (INEXACT)	Category name used on the Category statement. A Condition name used on ON CONDITION, SIGNAL, ENABLE, and DISABLE statements. Category name used on the Reprise statement.
XFER (TRANSFERT)	Keyword of the XFER statement.

Symbols

Relational Symbols: < <= > >= ==

Arithmetic Symbols: + - * / **

Comment Symbols /* */

Expression Symbols: () , and Relational and Arithmetic Symbols

Argument-list Symbols: Same as Expression Symbols

Assignment Symbol: <-

Text Symbols: &A &B ... &Z &- && < << > >> &;

SYMBOLS AND KEYWORDS BY STATEMENT

<u>STATEMENT</u>	<u>SYMBOLS AND KEYWORDS</u>
1. Assignment (Affectation)	; Expression Symbols, Assignment Symbol
2. CALL (APPELER)	; Argument-list Symbols, CALL
3. Category (Catégorie)	; F0, F1, F2, F3, F4, F5, F6, F7, F8, F9, RIGHT, WRONG
4. CHARACTER (CARACTERE)	; CHARACTER, BACKSPACE, INDEX, REVERSE INDEX, SUBSCRIPT, SUPERScript, LINEFEED, RETURN, HOME
5. CHECKPOINT (CONTROLE)	; CHECKPOINT
6. Conditional	Expression Symbols, IF, THEN, ELSE
7. DISABLE (INVALIDER)	; DISABLE, Condition names
8. DISPLAY (AFFICHER)	Text Symbols, DISPLAY, ON
9. DO (FAIRE)	; Expression Symbols, DO, TO, BY, WHILE
10. EDIT (EDITER)	; EDIT
11. ENABLE (VALIDER)	; ENABLE, Condition names
12. END (FIN)	; END
13. EXIT (SORTIE)	; EXIT
14. GOTO (ALLERA)	; GOTO, GO, TO
15. Header (Titre)	; Expression Symbols, PROCEDURE, FUNCTION, FONT, BLOCK, UNIT, EDIT, GRAPHIC, ENTRY
16. HOLD (RESERVER)	; HOLD
17. MAP (DESSINER)	; () , * , MAP, AT, BIN, HEX
18. Null (Vide)	;
19. ON CONDITION (SOUS CONDITION)	() ON CONDITION, and the Condition names
20. PLOT (TRACER)	; () , PLOT

<u>STATEMENT</u>	<u>SYMBOLS AND KEYWORDS</u>
21. PROCTOR (CENTRAL)	; PROCTOR
22. READ (LIRE)	; Expression Symbols, READ, KEY
23. RECORD (ENREGISTRER)	; RECORD
24. RELEASE (LIBERER)	; RELEASE
25. REPEAT (REPETER)	; Argument-list Symbols, REPEAT
26. Reprise	; Text Symbols, RETRY, REINF, Categories
27. RESPONSE (REPOSE)	; Expression Symbols, RESPONSE, ON, APPEND, CHARS, FONT, POSN, TIME, TYPE
28. RETURN (RETOUR)	; RETURN, CALLER, CONTINUE, ENTRY, REPEAT, RESTART
29. SET (FORCER)	; Assignment Symbol, SCOL, SCREEN, SEDIT, SFONT, SLANG, SPROC, SREC, STIME
30. SIGNAL (SIGNALER)	; SIGNAL, Condition names
31. STOP (ARRETER)	; STOP
32. SYN	; SYN
33. WRITE (Ecrire)	; Expression Symbols, WRITE, KEY
34. XFER (TRANSFERT)	; Argument-list Symbols, XFER
Comment (Commentaires)	Comment Symbols

Category names: F0, F1, F2, F3, F4, F5, F6, F7, F8, F9, RIGHT, WRONG

Categories: Category names, OTIME, RENTRY, RTYPE, UNREC

Condition names: Category names, ATTN, ENDCOURSE, ENDFILE, ENTRY, ERROR, INPUT, INSERTION, RESPONSE, RESTART

APPENDIX B

SYSTEM VARIABLES AND FUNCTIONS

(In those cases where the French Language Keyword differs from the English, it is shown in parentheses)

System Switch Variables

F0, F1, F2, F3, F4	unnamed category switches
F5, F6, F7, F8, F9	
FOTIME (FSURTEMPS)	overtime switch
FRIGHT (FEXACT)	Right category switch
FUNREC (FNONREC)	unrecognized response category switch
FWRONG (FINEXACT)	Wrong category switch

System Numeric Variables

ICRSE (ICOURS)	student course index
IDEVICE (IDISPO)	Student Device index
IERR	ERROR Condition index
ISESN (ISESS)	student session index
ITYPE	student device type index
LATENCY (DELA)	student latency of last response
MCRSE (MCOURS)	minutes this course (for student)
MSESN (MSESS)	minutes this session (for student)
NCRSE (NCOURS)	number of students in this Course
NFO, NF1, NF2, NF3	number of unnamed category occurrences
NF4, NF5, NF6, NF7	
NF8, NF9	
NOTIME (NSURTEMPS)	number of overtime category occurrences
NRESP (NREP)	number of responses
NRIGHT (NEXACT)	number of right category occurrences
NULL (VIDE)	null vector
NUNREC (NNONREC)	number of unrecognized responses
NWRONG (NINEXACT)	number of wrong category occurrences
PARS (PARET)	vector set at student registration

System String Variables

#QID (#ID)	queue of student names taking the Course
QUEUE (LISTE)	queue of pending Procedure names
REDITED (REDITE)	edited version of RESPONSE
RESPONSE (REPOSE)	last student response
SCOL (SCLAS)	collating sequence SET variable
SCREEN (SECRAN)	maximum screen size SET variable
SEDJT	default edit SET variable
SFONT (SJEU)	default Font SET variable
SID (IDET)	student identification code
SLANG	user language indicator
SNAME (NOMET)	student "nick" name
SPROC (SCENT)	proctor message header SET variable
SREC (SRES)	performance recording header SET variable
STIME (STEMPS)	default latency time SET variable
STUDENT (ETUDIANT)	student's full name
TIMED (TEMPSO)	time student started the Course
TIME1 (TEMPS1)	time student started the session
TIME2 (TEMPS2)	current time

System General Functions

ABS(x)	Absolute Value
CA(target, vec1, vec2, ...)	Compare Algebraic
CC(target, word, spec)	Compare Characters
CG(r, c, nr, nc, word)	Compare Graphic Area
CK(target, criterion)	Compare Keyletter
CN(target, tolerance, word)	Compare Numeric
COS(x)	Trigonometric COSINE
CP(target, word)	Compare Phonetic
INDEX(string, item) (Fr.INDICE)	Index of location
LEN(argument) (Fr.LONG)	Length of argument
LMATCH(string1, string2) (Fr.PDISC)	Location of match failure
LOG(x)	Base 10 logarithm
LOGE(x)	Natural logarithm
MAX(argument)	Maximum value
MIN(argument)	Minimum value
MOD(value, divisor)	Modulo value
NUMBR(string) (Fr.NUM)	String to numeric conversion
NUMSTR(string, item) (Fr.CHAINUM)	Number of occurrences of "item" in "string"
PICKN(mean, sd, num) (Fr.TIRNOR)	Normalized distribution random numbers
PICKU(low, high, num) (Fr.TIRUNI)	Uniform distribution random numbers
PMATCH(string1, string2) (Fr.CONC)	Percentage of match
RND(x) (Fr.ARR)	Round to integer
SIN(x)	Trigonometric SINE
SQRT(x) (Fr.RAC)	Square root
SUBVEC(vector, posn, len) (Fr.SSVEC)	Sub-vector construction
TAN(x) (Fr.TG)	Trigonometric TANGENT
TIME(time1, time2) (Fr.DUREE)	Time difference
TRUNC(x) (Fr.TRONC)	Truncate to integer
SHDN(string) (Fr.PASMIN)	Shift to lower case
SHUP(string) (Fr.PASMAJ)	Shift to upper case
STRING(n, chars, precision)(Fr.CHAINC)	Numeric to string conversion

System EDIT Functions

ALPHA	Extract alphabetic
MULSP (ESPMUL)	Remove multiple spaces
NUMBR (NUM)	Extract numeric
PUNC (PONC)	Remove punctuation
SHDN (PASMIN)	Shift to lower case
SHUP (PASMAJ)	Shift to upper case
STANDARD	Remove unrecognizable characters
SYMB	Remove all but alphabetic and numeric

System Graphic Functions

CHAR(character-string)	(Fr.CAR)	Display character string
CRCL(nx, ny, ...)		Display arc of circle
DRAW(nx, ny, ...)	(Fr.TRAIT)	Draw line(s)
ERASE(function calls)	(Fr.EFFACER)	Erase
FILL(nx, ny)	(Fr.REMPLIR)	Display filled rectangle
JOIN(x, y, ...)	(Fr.JOINDRE)	Draw line(s)
MOVE(nx, ny)	(Fr.DEPLACER)	Relocate "pen"
ORIG(x, y)		Reset origin
SETP(x, y)	(Fr.REPS)	Reset "pen"
XYXY(nx, ny, ...)		Draw horizontal and vertical lines

System Special Functions

SUBSTR(string, posn, len)	(Fr.SSCHA)	Sub-string construction
WORD(string, posn, len)	(Fr.MOT)	Sub-string construction

System Procedures

LINK(name, argument-list)	(Fr.LIEN)	Dynamic Invocation Procedure
CALC(variable-list, parameter-list)		Student Calculation Mode Procedure
EXEC(par5, par3, par4)		Load and Execute Procedure
name(argument-list)	(Fr.nom)	External System Invocation Procedure

APPENDIX C

179

CLASSES OF EXECUTION ERRORS

(IERR)

Internal Addressing

- GOTO label variable, location not found. 001
- GOTO indexed label, location not found. 002
- attempt to enter a DO-Block at location other than
DO-Block Header. 003
- RETRY on Reprise statement with no RESPONSE statement
in Unit. 004

External Addressing

- CALL external label not found. 101
- XFER external label not found. 102
- CALL LINK, external label not found. 103
- EDIT Edit Function not found. 104
- PLOT Graphic Function not found. 105
- General Function reference in expression, Function
not found. 106
- &F, Font Block or General Function not found. 107
- &G, Graphic Function not found. 108
- &V, Variable not available in Unit. 109
- REPEAT issued without Unit name, no prior Unit invoked. 110

Arithmetic

- Division by zero. 201
- Exponent overflow. 202
- Exponent underflow. 203
- Vector element access error (bad subscript). 204
- Unary operator on string. 205
- String in pseudo vector. 206
- Infix operator between string and vector. 207
- Negative value raised to fractional exponent. 208

Conversion

- attempt to convert a label variable. 301
- attempt to convert a numeric to character. 302

Variable Access

- attempt to write a protected system variable. 401
- access of undefined variable in Unit. 402

I/O Errors

- undefined "device". 501
- illegal KEY value. 502
- maximum record size exceeded 503
- device not operational. 504

180

(IERR)

Miscellaneous

- undecipherable contents in SET variable.

601

System Errors

- Student area size exceeded.
- Stack size exceeded.

901

902

IMPLEMENTATION LEVELS

The language described in this document is based on a set of requirements for a Computer Assisted Instruction language contained in "A Functional Specification for a Programming Language for Computer Aided Learning Applications," published by the National Research Council of Canada Associate Committee on Instructional Technology in Ottawa, July, 1972. In this document, four levels of implementation are identified, as follows:

- Level 2: This level should be considered the "full set" of the language. Implementation of the language at this level would include all of the proven features of the language. Any programs prepared using the subset of the language (either level 1 or level 0) would run on implementations at this level.
- Level 1: This level should be considered the basic subset of the language. It contains the minimum required processing capabilities and permits extensive use of input-output facilities.
- Level 0: This is the lowest level of implementation and should only be used in applications where input-output is limited to keyboard driven character display devices. All of the processing capabilities of level 1 will be included at this level, only the input-output handling capabilities will be reduced.
- Level X: This level should be considered an experimental level of implementation. Its prime purpose would be to test features which might be approved for implementation in the operational level(s) of the language. It would include all of the features of level 2 of the language as well as "experimental" features. It would normally be installed on systems being used to support the language. Any programs written in this level of the language must carry a warning regarding the experimental features required.

A Level 2 implementation of the language is described in this document, but level 0 or level 1 implementations may be provided at a particular installation. The NRC document referenced above defines the differences between implementations; the major differences are as follows.

Level 1 provides all of the facilities of level 2 except:

- General and Edit functions may not be defined. The mathematical functions such as SQRT, ABS, etc. are not provided; PICKN, CA, and CP functions are not provided; INDEX and SUBSTR are not provided.
- The extended RETURN statement is not permitted. The character swap string is not permitted on the EDIT statement. The WHILE clause and reverse indexing are not permitted on the DO statement.
- Logical and Course variables are not permitted.
- The functions of the &H control element are not supported. Devices other than terminals are not supported. Input time limit is not permitted.

Level 0 provides all of the facilities of level 1 (above) except:

- Font Blocks and Graphic functions may not be defined.
- The &B, &C, &D, &E, &F, &G, and &H control elements are not supported in the Display sub-language.
- Graphic input is not permitted.

	STATEMENT NAME	Proc	GnFn	EdFn	GrFn	FBlk	Unit	PROTOTYPE
1.	Assignment	*	*	*	*			l.h.s. <- r.h.s.;
2.	CALL	*						CALL name[(argument-list)];
3.	Category						*	category logical-expression;
4.	CHARACTER					*		CHARACTER character character-specification;
5.	CHECKPOINT	*						CHECKPOINT;
6.	Conditional	*	*	*	*			IF logical-expression THEN group1 [ELSE group2]
7.	DISABLE	*	*					DISABLE Condition;
8.	DISPLAY						*	DISPLAY [ON device] text
9.	DO	*	*	*	*			DO [variable = spec[,spec]...] [WHILE (logical-expression)];
10.	EDIT	*	*	*			*	EDIT edit-list;
11.	ENABLE	*	*					ENABLE Condition;
12.	END	*	*	*	*	*	*	END [location];
13.	EXIT	*						EXIT;
14.	GOTO	*	*	*	*			GOTO location; or GO TO location;
15.	Header	*	*	*	*	*	*	label : [function-type] routine-type [(argument-list)];
16.	HOLD	*	*					HOLD resource-list;
17.	MAP				*			MAP (nr,nc) [AT (r,c)] specification;
18.	Null	*	*	*	*			;
19.	ON CONDITION	*	*					ON CONDITION Condition action
20.	PLOT				*			PLOT Graphic-Function-references;
21.	PROCTOR	*						PROCTOR message;
22.	READ	*	*					READ (filename[,KEY=key]) read-list;
23.	RECORD	*						RECORD record;
24.	RELEASE	*	*					RELEASE;
25.	REPEAT	*						REPEAT [Unit-name[(argument-list)]];
26.	Reprise						*	action category text
27.	RESPONSE						*	RESPONSE [ON device] response-list;
28.	RETURN	*	*	*	*			RETURN [location];
29.	SET	*						SET set-variable <- set-list;
30.	SIGNAL	*						SIGNAL Condition;
31.	STOP	*						STOP;
32.	SYN	*	*	*	*	*	*	SYN name 'string's
33.	WRITE	*	*					WRITE (filename[,KEY=key]) write-list;
34.	XFER	*						XFER Procedure-name [(argument-list)];

 Proc = Procedure GrFn = Graphic Function
 GnFn = General Function FBlk = Font Block
 EdFn = Edit Function Unit = Instructional Unit
 * = statement permitted in that routine

This document is a cross-reference index between the Author's Guide and the Formal Specification Manuals of the Course Authoring Language.

The first column contains the indexed entries. The second column contains entries to the Author's Guide and the third column contains entries to the Formal Specification.

INDEXED ENTRY	AUTHOR'S GUIDE MANUAL	FORMAL SPECIFICATION MANUAL
ABS	76	153
absolute value	76	153
actions in condition blocks	149	90
addition operations	38	30
Algebraic comparison	98	146
ALPHA	92	162
alphabetics	2,15	12
alphameric comparison	95	148
ampersand character	20	15,21
APL	160	175
APPEND	91	104
APPEAL	151	93
arc of a circle	68	164
argument	3	5
argument list	4,41,45	33
arguments		
default value	46	35
input	45	34
output	46	34
two-way	46	34
arithmetic computations	160	171
arithmetic operator	4,32	23
Arithmetic Function	57	See General Function
arrays, one-dimensional	20	8
Assignment Statement	49	16
AS-IS mode	69	18
end of	117	50
start of	108	37
asterisk character	65	22
use of	30	22
AT	64	82
ATTN	146	85
audio units	88	
author	3	5
BACKSPACE	74	65
Basic Character Set	13,28	10,18
base ten value	18	155
begin AS-IS Mode	108	37
BIN	65	82
binary string	65	82
brightness	111	42
BY	50	70
CA	98	146
CALC	11,160	93
calculation mode	160	171

CALL	127	62
CALL Statement	127	62
CALLER	148	108
case shifting	81,82,93	162
Categorization Condition	146	86
category types	94	63
Category Statement	94	63
category switches	146	89
CC	95	148
CG	99	149
CHAR	69	163
CHARACTER	73	64
characters		
classifications	15	11
sequence of	5,131	8
character insertion	117	49
character set	12	10
basic	13	10
extended	14	11
character sequencing	131	8
CHARACTER Statement	73	64
example of	75	
character string	5	8
character-swap strings	92	74
character swap	93	74
CHECKPOINT	127	66
CHECKPOINT Statement	127	66
CK	100	150
CN	96	151
collating sequence	37	28
COMMENT	151	93
comment		
beginning of	27	18
end of	27	18
common resources	138	133
comparison function	94	63
component	3	5
Compilers	163,164	174
center text	113	44
CONDITION	142	108
Condition Block		
construction of	149	90
examples of	152	
execution of	149	
Conditional Statement	54,142	67
consecutive apostrophies	19	15
construction of statements	12	9
CONTINUE	148	108
control		
execution of	3	2
sequence of	3	5
transfer of	3,28	5
control character, changing of	114	45

control command		
characters	107	
construction	118	
elements	107	54
conventions	2	see guidelines
COS	78	154
Course variable	20	16
counters	24	139
CP	97	152
CRCL	68	164
Define Box	108	38
degrees of an arc	68	164
Device activation	109	39
digits	15	11
Direct Read	135	96
DISABLE	126	69
DISABLE Statement	126	68,122
DISPLAY	88	69
display circle	68	164
display conventions	118	51
display device	109	39
DISPLAY Statement	88	69
Display sub-language	89	36
division operation	38	156
DO	50	70
DO-Block	50	70,119
DO While	51	70
DRAW	68	164
E-notation	3	5
EDIT Function	60	162
construction of	60	124
example of	61	
header statement	61	
invoking of	60	
EDIT	92	74
EDIT Statement	92	74
elapsed time	81	138
Elemental Graphic Functions	66	163
ellipses	2	60
ELSE	54	67
ENABLE	126	75
ENABLE Statement	126	75
END	56	76
END Statement	56,88	76,120
ENDCOURSE	147	86
ENDFILE	144	86
ENTRY	144,148	87,108

Entry Procedure		
construction of	158	
example of	158	
header statement	158	
invoking of	157	
ERROR Condition	43,46,145	87
ERASE	68	165
Erase screen area	110	40
execution sequence	3	5
EXIT	130	77
EXIT Statement	130	77
exponent	3	5
exponentiation	34	25
exponentiation operator	34	25
Expressions		
numeric	34	25
string	34	26
vector	37	28
Extended Character Set	14,28	11
access to	117	49
Extensor	117	50
external compilers	163	174
external label	30	21,118
FILL	69	165
F0, F1...F9	22	63,136
FONT	91	105
Font Block	71	79
construction of	72	125
header statement	72	
invoking of	71	
Format control, resetting of	114	50
FOTIME	22	137
FRIGHT	22	137
FUNCTION	30	79
Functions		
Arithmetic	57	153
Edit	92	162
Font Block	71	79
General	57	153
Graphic	62	163
invoking of	31	127
labelling	30	118
Pseudo	82	168
string	81	160
function calls	69	165
Function reference	110	41
FUNREC	23	137
FWRONG	22	137

General Function		
construction of	58	123
example of	59	
header statement	58	
invoking of	57	
General Procedure		
construction of	154	
example of	155	
header statement	154	
invoking of	153	
GLOSSARY	151	93
GOTO	55	78
GOTO Statement	55	78
graphic	4	6
GRAPHIC	64	79
graphic character	73	64
Graphic Comparison	99	149
Graphic Function	62,66,111	163
construction of	63	124
example of	70	
Header Statement	64	
invoking of	62	
graphical presentation	107	163
graphic symbol redefinition	71	125
guidelines		59
hanging indent	114	45
Header Statement		79
Edit Function	61	
Entry Procedure	157	
Font Block	72	
General Function	58	
Graphic Function	64	
General Procedure	154	
Unit	87	
HELP	151	93
HEX	65	82
highlighting	111	42
HOLD	140	80
HOLD Statement	140	80
HOME	75	65
horizontal lines	68	167
hyphen, use of	121	
ICRSE	23	137
identification codes, student	26	143
IDEVICE	23	137
IERR	23	137
IF	54	67
implementation	4	6
INDEX	74	154
Index Function	76	154

index number	23	137
indexed label	31	22
indexed literal label	29	20
input arguments	45	34
INPUT	147	87
Input type	23	138
INSERTION	144	88
installation, physical	34	6
instructional material, presentation of	7	2
instructional		
specifications	7	3
transaction	8	2
unit	8,87	3
intentional sign-off	163	134
internal labels	28	119
Interpreters	163	174
interrupt processing	146	85
ISESN	24	138
ITYPE	23	138
JOIN	67	166
joining coordinates	67	166
keep on page	113	44
KEY	135,136	96,113
keyword	4	6,18
Keyletter Comparison	100	150
label	30	21
external	30	19,21
literal	20,29	20
indexed	31	22
indexed literal	29	20
internal	28	19
label variables	17,20,30	15,21
labels	28	19
Language		
element of	12	7
purpose of	1	1
LATENCY	24	138
LEN	76	154
letters		
lower case	15	12
upper case	15	11
letter transposition	97	152
line delimiters	27	18
LINEFEED	75	65
line spacing	113	44
LINK	160	44,173
list, argument	4,45	7,34

literal label	20	20
numeric	18	14
string	19	15
vectors	21	13
literal	4	6,13
label	29	20
number	3	5
numerics to spaces	92	162
LMATCH	77	155
location, destination	29	20
locked resources	140	80,81,99
locked-in	9	
locked out	139	80
LOG	77	155
LOGE	77	155
logical operators	32	23
lower case letters	15	12
MAP	64	82
MAP Statement	64	82
mantissa	9	5
manual		
purpose of	1	1
use of	11	2
margin indent	112	43
margin setting	112	43
MAX	77	155
MCRSE	81	138
MIN	77	155
MOD	78	156
modularity	9	
MOVE	67	166
MSESN	81	138
MULSP	92	162
multiple line spacing	115	46
name, logical	28	19
NCRSE	81	138
nested statements	54	67
new page	113	44,53
new paragraph	114	45
new screen	113	44
NFO, NF1,...NF9	24	139
nick name	26	43
non-digit to spaces	92	162
NOTIME	25	139
NRESP	24	139
NRIGHT	24	139
Numeric Comparison Function	96	151
Numeric expression, example of	40	25
Numeric literals	18	14
numeric values	18	8

numeric variables	18,106	14,137
NULL	81	140
Null Statement	55	84
null string	19	14
NUMBR	78,92	156,162
NUMSTR	78	156
NUNREC	25	140
NWRONG	24	140
offset	114	46
ON	126,142	69,85,104
ON CONDITION Statement	142	85
operators		
arithmetic	4,32	7
logical	32	7,23
optional hyphen	118	50
order of execution in Unit	103	
order of calculation in expression	38	29
ORIG	67	166
OTIME	146	63,88
output arguments	46	34
over printing	110	40
paragraphs	114	45
parameter	4	7
PARS	25	140
pause	116	49
pending	5	7
Phonetic Comparison	97	152
physical end	56	76
physical location	34	6
picture graphics	64	82
PICKU	79	157
place-holders	5	5,59
PLOT	66	94
PLOT Statement	66	94
PMATCH	79	158
Point graphic	63	82
POSN	90	105
prefix operators	34,37	25,28
PROCEDURE		79
Procedure routines, labelling	30	21
PROCTOR	130	95
PROCTOR Statement	130	95
prototypes	2,9	
arrangement of	2	
pseudo vectors	21,42	17,32
PUNC	93	162
punctuation to spaces	93	162
purpose of manual	1	1

QUEUE	26	141
random numeric selection	78	157
READ	135	96
Read Pointer	134	96
READ Statement	135	96
record deletion	136	113
RECORD	131	98
RECORD Statement	131	98
REDITED	26	141
refresh display	113	44
REINF	101	102
RELEASE	141	99
RELEASE Statement	141	99
released resources	140	99
removing spaces	92	162
RENTY	101	102
REPEAT	148	108
REPEAT Statement	128	100,121
Reprise Statement	101	102
resource access	138	133
resource lockout	140	80
resource list	140	80
resource sharing	138	133
reserved symbols	16	12
RESPONSE	25	141
Response Condition	145	89
RESPONSE Statement	89	104
RESTART	148	108
major	166	66,77,135
minor	166	66,135
RESTART	144	89
Restart points	127	66
RETRY	101	102
RETURN	55,73	65,107
RETURN Statement	55	107
REVERSE INDEX	75	65
RIGHT	94	63
right justification	112	43
RND	79	158
roots	34	25,158
rounding values	79	158
routine	5	7
construction of	28	123
types of	7	116
routine end	56	76
RTYPE	145	63

SCOL	131	142
SCREEN	132	142
screen position	111	42
SEDIT	132	143
semi-colon	55	84
Sequential Read	135	96
Sequential Write	136	113
SET	131	109
SETP	67	166
SET Statement	131	109
SFONT	132	143
SHDN	81,93	159,162
SHUP	82,93	160,162
SID	26	143
SIGNAL	125	110
SIGNAL Statement	125	110
SIN	78	157
sine	78	157
skip lines	113	44
SLANG	132	143
slide projector	88,109	
SNAME	26	143
space character	15	12
special characters	120	52
specifications, instructional	7	2
spelling errors	97	152
SPROC	133	144
SQRT	79	158
square root	79	158
SREC	133	144
STANDARD	93	163
statement	5	7
Assignment	49	61
CALL	127	62
Categorization	94	63
CHARACTER	73	64
CHECKPOINT	127	66
Conditional	54	67
components	12	9
construction of	12,27	18
DISABLE	126	68
DISPLAY	88,107	69
DO	50	70
EDIT	92	74
ENABLE	126	75
END	56,88	76
End of Display	118	50
EXIT	130	77
Extended Return	147	
GOTO	55	78
Header	28,30	79
HOLD	140	80
labelled	20,28	19
MAP	64	82

Null	55	84
ON CONDITION	142	85
PLOT	66	94
PROCEDURE	125	95
PROCTOR	130	95
READ	135	96
RECORD	131	98
RELEASE	141	99
REPEAT	128	100
Reprise	101	102
RESPONSE	89	104
RETURN	55	107
SET	131	109
SIGNAL	125	110
STOP	129	111
SYN	167	112
WRITE	136	113
XFER	128	115
statement delimiter	56	76
start of text	109	39
TIME	133	144
STOP	129	111
STOP Statement	129	111
STRING	82	160
string		
alphabetic characters	6	8
characters	5,6	8
digits	6	8
end-of	6	8
non-alphabetic	6	8
null	19	14
string expression, example of	40	26
string literal	19	15
string variables	19,106	14,141
strobe effect	111	42
STUDENT	5,26	8,144
familiar name	26	143
full name	26	143
identification code	26	143
nick name	26	43
usual name	26	43
SUBSCRIPT	75	65
subscripts	19,21,120	15,28
SUBSTR	83	168
SUBVEC	80	161
SUPERSCRIP	75	65
superscript	120	52
switch variables	17,106	13
switch expression, example of	40	
SYMB	93	163
symbols	2,15	12
reserved	16	12
two-character	16	12

symbols to spaces conversion	93	162
SYN	167	112
SYN Statement	167	112
example of	168	
synonym	167	
system	5	8
system comparison functions	95	148
System Functions	76	146
system switches	22	137
numeric	23	138
tabulation	115	47
TAN	78	159
tangent	78	159
Terms		
Argument	3	5
Author	3	5
Component	3	5
Control	3	5
Definition	3	5
E-notation	3	5
Font	3	6
Graphic	4	6
Installation	4	6
Invoke	4	6
Implementation	4	6
Keyword	4	6
Literal	4	6
Operator	4	7
Parameter	4	7
Pending	5	7
Routine	5	7
Statement	5	7
String	5	8
Student	5	8
System	5	8
Vector	5	8
Word	6	8
text conventions	119	52
text display	88	39
text emphasis	111	42
text justification	112	43
textual presentation	107	36
THEN	54	67
TIME	80	106,159
TIME0	85	145
TIME1	85	145
TIME2	85	145
time difference calculations	80	159
time limit	146	88
TO	50,55	70,78
transactions	8	2
trigonometric functions	78	154,157,171

TRUNC	80	159
two-way arguments	46	34
TYPE	91	105
underlining	115	47
underscore, use of	121	
unintentional sign-off	166	134
Unit routines		
construction of	103	122
example of	122	
labelling	30	
Unit System Variables	106	
UNREC	102,146	63,89
upper case letters	15	11
user language	132	143
value conversion	41	31
variables	7,17	13
course	7,20	16
insertion of	116	48
label	17,20,30	15,21
name	3,21	5,13
numeric	18	14
numeric pseudo	81	138
string	19,106	14
string pseudo	85	145
switch	17,106	13
system	7,22	136
undefined	57	
vector	5,20	8
literal	21	17
name	21	17
null	20	16
pseudo	21,42	17,32
variables	20	16
vector expressions	37	28
example of	40	
vector graphics	66	
vector passing	45	33
vector recomposition	80	161

wait	116	49
WHILE	50	70
window effect		83
word	6	8
WORD	84	169
word boundary	84	8,50
Write Pointer	134	113
WRITE	136	113
WRITE Statement	136	113
WRONG	94,101	63,102

XFER	128	115
XFER Statement	128	115
XXXY	68	167

#QID	26	141
&A	108	37
&B	108	38
&C	109	39
&D	109	39
&E	110	40
&F	110	41
&G	111	41
&H	111	42
&I	112	43
&J	112	43
&K	113	44
&L	113	44
&M	113	44
&N	113	44
&O	114	45
&P	114	45
&Q	114	45
&R	114	46
&S	115	46
&T	115	47
&U	115	47
&V	116	48
&W	116	49
&X	117	49
&Y	117	50
&Z	117	50
&-	118	50
&;	118	50